SKILL CAPTURE IN FIRST-PERSON SHOOTERS

A thesis submitted to the University of Manchester for the degree of Doctor of Philosophy in the Faculty of Engineering and Physical Sciences

2016

By
David Buckley
School of Computer Science
Contents

Abstract 14
Declaration 15
Copyright 16
Acknowledgements 17

1 Introduction 18
1.1 What are skill and difficulty, and how do they relate to each other? 18
1.2 Why are these concepts so important in video games? 19
1.3 How do we currently match difficulty to skill? 20
1.4 How can we do it instead? 21
1.5 What does this research offer? 22
1.6 How is this research laid out? 23

2 Background 25
2.1 A short taxonomy of skill 25
2.1.1 The skill, performance and difficulty triangle 25
2.1.2 Skill capture 28
2.1.3 Bloom’s taxonomy 28
2.1.4 Conroy’s model of skill 29
2.1.5 Player Modelling: accounting for different kinds of people 30
2.2 Dynamic Difficulty Adjustment (DDA) 31
2.2.1 Commercial examples 31
2.2.2 DDA in the literature 33
2.2.3 Shortcomings 34
2.3 Skill capture in the literature 34
3 Skill Prediction in a Multiplayer Game

3.1 Problem statement ................................................. 61

3.2 Experimental setup ................................................. 62
  3.2.1 The Log File ................................................. 63
  3.2.2 Data collection ................................................. 64
  3.2.3 Questionnaires ................................................. 67
  3.2.4 Data Distribution ............................................... 69

3.3 Measuring skill .................................................. 71
  3.3.1 Rank .......................................................... 71
  3.3.2 Score ........................................................ 74
  3.3.3 TrueSkill ...................................................... 78
  3.3.4 Self-reported measures ....................................... 78
  3.3.5 Community measures ......................................... 82
  3.3.6 Summary ...................................................... 84

3.4 Feature extraction ............................................... 84
  3.4.1 Hardware: keyboard, mouse movement and clicks ........ 87
  3.4.2 Type: event frequency, complexity and kinetics .......... 88
  3.4.3 Context: Free and Dependent .............................. 89
  3.4.4 Player learning .............................................. 89

3.5 Prediction ........................................................ 91
  3.5.1 Predicting a skill category .................................. 92
  3.5.2 Predicting skill measures ................................... 93
  3.5.3 Prediction convergence rate .................................. 95
3.5.4 Predicting affective state and preference .............. 97
3.6 Summary ................................................. 97

4 Automatic Difficulty Selection ................................................. 99
4.1 Problem statement .............................................. 100
4.2 Project:Blue Room: The test-bed ................................. 103
  4.2.1 Levels .................................................. 104
  4.2.2 Difficulty on Project:Blue Room .............................. 109
  4.2.3 Experiment setup ........................................... 111
  4.2.4 The adapted Game Expertise Questionnaire ................. 115
  4.2.5 Data distribution .......................................... 116
4.3 Measuring difficulty and performance ......................... 118
  4.3.1 Performance ............................................. 119
  4.3.2 Difficulty .................................................. 124
4.4 Predicting skill .................................................. 127
  4.4.1 Features ................................................. 128
  4.4.2 Prediction ................................................... 129
4.5 Applying skill capture to difficulty selection ............... 133
  4.5.1 Experimental setup ......................................... 133
  4.5.2 Data distribution .......................................... 136
  4.5.3 Results ..................................................... 137
4.6 Summary ......................................................... 141

5 Conclusions ................................................................. 145
5.1 Summary of results .................................................. 145
5.2 Discussion ......................................................... 146
  5.2.1 Multiplayer ............................................... 146
  5.2.2 Single-player .............................................. 151
  5.2.3 A psychological view on Conroy’s framework for skill ... 155
  5.2.4 Potential application ........................................ 158
5.3 Research output ..................................................... 160
5.4 Future work ........................................................ 162
5.5 Concluding remarks ................................................. 166

Bibliography ................................................................. 167

A The Game Expertise Questionnaire .................................... 178
B  Red Eclipse Features  180

C  Project:Blue Room Features  193
  Word Count: 39,990
List of Tables

3.1 Details for each map used in the experiment. .................. 64
3.2 The metadata for each game. ................................. 65
3.3 Game events recorded by Red Eclipse. ....................... 66
3.4 A summary of the skill measures in Red Eclipse. ............. 72
3.5 The score groups separated by player score ($\bar{s}$). .......... 77
3.6 Magnitude of correlation (Spearman’s $|\rho|$) of all skill measures, where values > 0.9 and < -0.9 are highlighted. Skill measure notation is defined in Table 3.4 for reference. ................. 80
3.7 Feature groups used within this research ..................... 85
3.8 The top ten features ranked according to their correlation to player score ($\bar{s}$). The entire list of features can be found in Appendix B. 86
3.9 How each game was classified for a random forest trained to predict groups of player score. Each row indicates how the examples were classified. In other words, there were 5 Novice players misclassified as Intermediate. ................................. 93
3.10 Test accuracy (%) of random forest models trained to predict player responses based on player input to the game. Standard deviations are presented in brackets. ....................... 97

4.1 The events recorded in Project:Blue Room. .................. 114
4.2 A summary of performance measures in Project:Blue Room. . . . 123
4.3 The classifications for a random forest trained to predict reported expertise ($e$) for a portion of the data set, presented in a confusion matrix. The numbers presented here are answers to the Likert questionnaire regarding expertise. As before, rows indicate actual value (two players with $e = 1$ were misclassified). ....................... 130
4.4 The final 16 features used by the SVR model to predict reported expertise, $e$, ordered by decreasing importance. The correlation given represents the model’s accuracy after removing the given feature in the reverse order given. The complete list is presented in Appendix C.
List of Figures

1.1 The difficulty options given to players in *Doom*. . . . . . . . . . . . 20

2.1 How the concept of skill is connected to similar, relevant concepts. 26

2.2 How the player’s skill, the difficulty of the content affect the chal-

lenge, and subsequently the player’s performance. . . . . . . . . . . 27

2.3 A sample of the Game Expertise Questionnaire for the Logic/Puzz-

le genre. Three questions are asked and the player is asked to
circle any games played. The full questionnaire can be found in
Appendix A. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42

2.4 Two possible decision boundaries for the same data. Both classify
the existing data in the same way. Line A, however, has more room
for error given any new points. . . . . . . . . . . . . . . . . . . . . 46

2.5 The dimension $x$ cannot be classified by a single linear classifier.
However, transforming it into $x^2$, the data can be linearly separated. 47

2.6 In linear regression, Line A is chosen to minimize the distance
between it and the surrounding points. . . . . . . . . . . . . . . . 48

2.7 Projecting the data onto vector A would keep the majority of the
data’s variance. However, transforming the data onto B allows
better differentiation between the two classes. . . . . . . . . . . . 49

2.8 The dimension $x_1$ has a greater between-class scatter, but greater
within-class scatter, meaning $x_2$ is more suitable. . . . . . . . . 50

2.9 A single signal composed of four sine waves of different frequencies
and amplitudes. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56

2.10 The signal in Figure 2.9 broken down into its composite parts. . . 56

3.1 The overall format of the experiment. . . . . . . . . . . . . . . . . 65
3.2 The number of games played by each player. Games highlighted by the dashed box are those used in this research. Colors indicate which maps each player played. ............................................. 69

3.3 The number of times each map was played, overlaid by the number of times played by each group in \( f \) (FPSs Played). ................. 70

3.4 As in Figure 3.3, the number of games played on each difficulty, with additional grouping over \( f \) (FPSs Played). ................. 70

3.5 This Tukey box plot [1] presents the performance measure, rank \((r)\) and skill measure \((\bar{r})\) for every game, grouped by the game’s map. Here, a lower value indicates higher performance. On average, players performed slightly worse on the maps Foundation and Wet. 73

3.6 Rank \((r)\) and player rank \((\bar{r})\) presented in the same notation as Figure 3.5, grouped instead by difficulty. Harder difficulties (e.g. 90–100) led to much lower performance. ....................... 73

3.7 As in Figure 3.5, but using performance and skill metrics score \((s)\) and player score \((\bar{s})\) respectively. Higher \(s\) indicates higher performance. .................................................. 75

3.8 As in Figure 3.6, but with score \((s)\) and player score \((\bar{s})\). As with \(r\), \(s\) was lower for higher difficulties. ....................... 75

3.9 Score \((s)\) for each player, ordered by the players’ mean scores \((\bar{s})\). Games highlighted in red indicate unexpectedly high values of \(s\). 76

3.10 Distribution of player ranks \((\bar{r})\) compared to the distribution of player scores \((\bar{s})\). ................................. 76

3.11 The correlation, \(|\rho|\) between \(\bar{s}_i\) and final player score \((\bar{s})\), where \(\bar{s}_i\) is calculated by averaging score \((s)\) over the first \(i\) games for each player. .................................................. 77

3.12 The distribution of scores \((s)\) for each skill group. The number of players for each group is given in Table 3.5. ....................... 78

3.13 Average TrueSkill estimates \((T)\) before each game for each score group. From bottom to top, each dotted line represents a bot difficulty; 40–50, 60–70, 50–60, 70–80, 80–90 and 90–100. ........... 79

3.14 Convergence of the average TrueSkill \(\sigma\) for each score group over time. ............................................................ 79
3.15 Relationship between the skill measures TrueSkill estimate ($T$), player score ($\bar{s}$) and player rank ($\bar{r}$). Highlighted players have much higher relative values of $\bar{s}$ than their equivalent $T$ or $\bar{r}$.

3.16 The distribution of score ($s$) and player score ($\bar{s}$) for each hours played ($h$) group.

3.17 The distribution of score ($s$) and player score ($\bar{s}$) for each FPSs played ($f$) group.

3.18 How player rank ($\bar{r}$) is related to average kills-per-death ($\bar{k}$).

3.19 The relationship between player accuracy ($\bar{a}$) and player rank $\bar{r}$.

3.20 Pearson correlation coefficient for each feature to player score ($\bar{s}$), grouped by feature group and ordered by correlation. Dotted lines indicate correlation of $\pm 0.6$.

3.21 Cumulative average score ($s$) over several games for each score group. The cumulative average score is calculated on the scores of all games up to the given game number.

3.22 Cumulative average value for a feature over several games for each score group. For each game number, the average is taken over all games up to that point.

3.23 The correlation of the cumulative averages to the final average for score ($s$) and a feature of player input.

3.24 Mean accuracy of random forest trained to predict a player’s score group using different feature groups. Error bars indicate standard error of each model.

3.25 Mean accuracy of random forest trained to detect Novice players using different feature groups. The baseline is from always guessing the player is not in the Novice category (but in one of the others instead).

3.26 Performance (Spearman’s $|\rho|$) of models trained to predict player score ($\bar{s}$). Baseline indicates mean $|\rho|$ between score ($s$) and $\bar{s}$.

3.27 Relationship between predicted player score ($\hat{s}$) and predicted player accuracy ($\hat{a}$), coloured by score group.

3.28 How fast a classification model is able to distinguish between two groups of players (Intermediate or worse, and Skilled or better). Dotted line indicates mean accuracy guessing the majority class.
3.29 How fast a regression model is able to predict player score \((\bar{s})\).
Baseline indicates mean correlation of the current score at \(t\). . . . 96

4.1 How the first level, Hazard, was laid out. Key elements introduced to the player have been highlighted. Points where the player located the weapons are also shown. . . . . . . . . . . . . . . . . . . . 105

4.2 At the end of Hazard, the player was asked to shoot at red targets, shown here. Destroying two targets would allow them to progress. 105

4.3 A puzzle in the level Introduction. Players were required to use the given crates (brown) to get to the distant area (orange). . . . 106

4.4 The puzzle shown in Figure 4.3 was changed according to the difficulty setting. Static blocks were introduced for easier difficulties to add simplicity. . . . . . . . . . . . . . . . . . . . . . . . . 106

4.5 The layout of the third level, Combat. Approximate locations of enemies have been labelled. These can change according to difficulty. The room labelled ‘waves’ has three waves of enemies, each wave increasing in number and difficulty of opponents. . . . . . . 108

4.6 The final room in Combat. A set number of enemies spawn behind random doors in three waves. . . . . . . . . . . . . . . . . . . . . 109

4.7 There were a variety of routes available to players for the level Strategy, some of which have been highlighted here. Each of the routes presented a different challenge. . . . . . . . . . . . . . . . . 110

4.8 The level, Arena. The ammo points labelled were the predetermined locations for the RPG ammo. The spawn rooms are where waves of enemies would appear at regular intervals. . . . . . 111

4.9 An ammo point lit with a green light used to guide players. The final boss is also visible here on the left. . . . . . . . . . . . . . . . 112

4.10 Average game progress, \(p_G\), and time playing \(t_G\), for participants who took part in-house against those who took part online. . . . 117

4.11 The distribution of answers for the number of first-person shooters played, \(f\). Values on the x-axis indicate the questions asked of the participants. The in-house participants have been visualised separately. . . . . . . . . . . . . . . . . . . . . . . . . 118

4.12 The distribution of reported expertise, \(e\), for this experiment. . . . 119

4.13 The distribution of average hours played per week in the last six months, \(h\), for this experiment. . . . . . . . . . . . . . . . . . . . 119
4.14 The relationship between two performance measures, game progress, $p_G$ and completion time, $t_C$, to difficulty, $d$. Standard error is indicated for each measure ($\frac{\sigma}{\sqrt{N}}$). .................................................. 121

4.15 The relationship between the adjusted performance measures (used as proxies of skill) to difficulty, $d$. Standard error is given for each window. ............................................................ 122

4.16 Average game progress, $p_G$, for each expertise group. There were no players in this experiment that reported having an expertise of $e = 0$. .......................................................... 123

4.17 Average game progress, $p_G$, for each FPSs played group, $f$. There were no players that recorded $f = \text{None}$. ......................... 124

4.18 The number of total attempts for the level Introduction against the number of complete attempts, grouped by difficulty ranges, $d$. 125

4.19 The proportion of completed attempts over difficulty, $d$, for the level Introduction. As difficulty increases, attempts are less likely to be completed. ..................................................... 126

4.20 The proportion of completed attempts over difficulty, $d$, for the level Combat. Difficulty drastically increases for values of $d > 0.4$. 126

4.21 Cumulative probability of completion for two levels, Introduction and Arena, demonstrating selecting difficulty values, $d$, for different skill measures, $s$. Mapping $s = 0.5$ and $s = 0.8$ produces $d = 0.458$ and $d = 0.745$ respectively for Introduction, and $d = 0.205$ and $d = 0.478$ for Arena. In each case, the cumulative probability has been normalised to between 0 and 1. ................................. 127

4.22 The first two dimensions of an LDA model trained on reported expertise, $e$. ................................................................. 130

4.23 How performance of an SVR model trained to predict reported expertise, $e$, improves with the removal of given features. ....... 131

4.24 Two features grouped by reported expertise, $e$; numdiffkeys, which describes player input, and progress-maxtime, which describes the speed at which they complete the first challenge. .......... 133

4.25 The predicted values from the SVR model trained to predict reported expertise, $e$, for each game. ................................. 134

4.26 The overall format of the final experiment. There are five levels in total, each which had an associated question about fun and difficulty. 136
4.27 The distribution of the number of years played by each participant in Phase II of Project:Blue Room. .......................... 137
4.28 The distribution of the number of first-person shooters played by each participant in Phase II of Project:Blue Room. .................. 137
4.29 The distribution of self-reported expertise by each participant in Phase II of Project:Blue Room. .......................... 138
4.30 The distribution of hours per week played by each participant in Phase II of Project:Blue Room. .......................... 138
4.31 The difficulty assigned to each expertise group, \( e \), for the second level. ................................................. 139
4.32 The average progress made per second for each of the expertise groups, \( e \) in each of the test groups. ......................... 140
4.33 The average progress made per second for each of the test groups. 140
4.34 The reported fun and difficulty for the whole game for each test group. ................................................. 141
4.35 The time for each game shown against the time each participant thought they had been playing. ......................... 141
4.36 How much each participant overestimated or underestimated the time they had been playing. “Flow error” is the reported time, given in Figure 4.35, minus the actual game time. A positive number indicates an overestimation, and a negative number indicates underestimation, or stronger flow. .......................... 142

5.1 The relationship between the action-perception model and Conroy’s model, coloured........................................ 156

A.1 The original Game Expertise Questionnaire. ......................... 179
Abstract

SKILL CAPTURE IN FIRST-PERSON SHOOTERS
David Buckley
A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy, 2016

The current models of skill in video games make one of two impositions on players: either to provide an estimate of their own skill, or complete several games before they can be properly assessed. However, in order to experience the most enjoyment and greatest sense of immersion, players need to play against the right difficulty. In order to assign the appropriate difficulty, the player’s skill must first be captured accurately and quickly, before the player gets frustrated or bored. Rather than relying on game results that need to be averaged over several games, this thesis proposes predicting a player’s skill from their behaviour within the first game. In order to do this, we explore methods for measuring skill in both a multiplayer and single-player game and methods for extracting appropriate information from the player’s behaviour. The resulting predictions can then be used to automatically assign an appropriate difficulty to the player. In a multiplayer environment, we first demonstrate that a player’s final rank can be predicted within the first 30 seconds of a game with a correlation of over 0.8. This process is transferred to a single-player first-person shooter, where our model is shown to assign difficulties comparable to a player’s own assessment of their skill within the first 30 seconds of a campaign. We argue that these methods for capturing skill in a first-person shooter are transferable to other genres, and have the potential to improve difficulty selection systems.
Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.
Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=487), in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see http://www.manchester.ac.uk/library/aboutus/regulations) and in The University’s policy on presentation of Theses.
Acknowledgements

Thanks go to the participants at the University of Manchester and Salford for assisting with experiments. Thanks also go to Stephen Mitroff, who provided the Game Expertise Questionnaire, and David Conroy, for his correspondence on his skill model.

This work was supported by the Engineering and Physical Research Council [EP/I028099/1].
Chapter 1

Introduction

1.1 What are skill and difficulty, and how do they relate to each other?

One thing that everyone has in common is that they have had to learn how to do something at some point in their life. Whether this was an instrument, sport, or how to speak their mother-tongue, everyone has had the opportunity to evaluate and compare their own abilities to others. Some people are more skilled than others, and some tasks are harder than others. Such is the way of life.

In order to present our ideas, we need to be using consistent definitions of skill and difficulty. In this research, we define skill as someone’s average performance, where performance is how well they do at something. So a skilled person, on average, will do better at a task than an unskilled person. In contrast, difficulty is the objective measurement of a task that determines the level of challenge that a player of a particular skill will experience.

So what is challenge? Some people use challenge and difficulty interchangeably, but we have chosen to use challenge to reflect the difference between a player’s skill and the static difficulty of the task, i.e. greater difficulty results in greater challenge.
1.2 Why are these concepts so important in video games?

The problem of finding the right level of challenge for someone is one that has been around for as long as the notion of learning. When something is too hard, people can get frustrated, and when it’s too easy, they don’t learn anything, or may not be stimulated enough. So we need to find a balance between the two.

Csíkszentmihályi offers a model of flow, which states that when the difficulty of the game matches the skill of the player (i.e. the challenge is right), the player experiences the most enjoyment [2, 3]. Given that people play video games for enjoyment, maximising that enjoyment is important for both the players and the developers selling the games.

However, the number of different kinds of people playing video games is vast. Traditionally, video games were dominated by the 18 - 25 male demographic. Tailoring to this demographic became relatively straightforward, and could be done with minimal effort. This was somewhat exclusive, and created stigmas around the pastime. However, as these stigmas have faded, the demographics have widened, and a much more diverse gamer base has blossomed. As this occurs, games have become more difficult to tailor. The number of different types of people increases, and the percentage of whom a particular video game might appeal to decreases.

In addition to a rich storyline and entertaining gameplay, developers must also adjust the skill so that it is accessible to these newer audiences; players who have less experience, time or less motivation to accomplish difficult tasks. This has resulted in some ‘dumbing down’ of games which underestimate their players, a fact that ‘traditional gamers’ have berated the games industry for [4].

Correctly tailoring this difficulty so that novice players and experts can get the maximum enjoyment out of it requires a lot of work on the part of the developer. Most games will also offer several different difficulties so that different player types and skills can be accounted for. But how is a player supposed to understand which difficulty they should select without prior experience of the game? This task of matching the appropriate difficulty to the appropriate player is one that has been studied for the past couple of decades, and that this thesis will attempt to explore.
CHAPTER 1. INTRODUCTION

1.3 How do we currently match difficulty to skill?

Traditionally, video games have asked a player how hard they would like their game to be. *Doom* [5], for instance, would ask players to select between five different difficulties, as seen in Figure 1.1. The game’s settings would be changed accordingly. For example, enemies on the easiest option, *I’m too young to die*, would only deal half the normal damage to players. Other options would vary the number of enemies the player faced.

A problem with this is that a player would have to know what the developer meant by each difficulty. One developer’s ‘hard’, for instance, may be too easy for another. One of the criticisms of the 2004 game *Half-Life 2* [6] was that it was less challenging than its 1998 predecessor, *Half-Life* [7, 8]. How are players therefore meant to understand what a developer means by ‘easy’ or ‘hard’?

One solution to this problem was Dynamic Difficulty Adjustment (DDA) [9]. DDA sought to automatically change the difficulty on-the-fly according to how well the player was doing. There is plenty of research in this field that attempts to more accurately model each player’s performance over time. More importantly, there is research that explores how challenge should be assigned over the course of the whole game. For instance, should the game get steadily more challenging throughout the game, or even ease off at times?

However, a major downfall with DDA is that players start competing against the DDA engine rather than the game itself. If a player is rewarded for poor performance (by making the game easier when they do particular things), the
player is likely to attempt to continue performing poorly. Players are even punished for doing well. If the player successfully defeats a horde of enemies they did not think they could defeat, they are either punished by making the next group harder, or feel disappointed if they discover the enemies were designed to be easier in response to their poor performance.

These difficulties that arise with regards to challenging a player are the main reasons behind this research. How can we accurately model a player’s skill within a video game, and how can we use this knowledge to select the appropriate difficulty for the player?

It is worth noting that these questions don’t solely apply to video games, and nor do they solely apply to games in general. Any method of skill acquisition, such as language learning or even playing a musical instrument, faces the same problems. How do you select a difficulty for the user to aid them to learn the fastest while not overwhelming them?

### 1.4 How can we do it instead?

There is a relationship between skill, difficulty and performance that was already highlighted in Section 1.1. The problem with player-based difficulty selection is that players don’t understand the difficulty, and are often poor judges of their own skill (à la the Dunning Kruger Effect [10]).

DDA methods attempt to address this by taking the choice away from the player. Unfortunately, the developer does not know what the player’s skill is, and has to rely on individual measures of performance to tailor the difficulty. A truly accurate measure of skill would require too much time playing. Moreover, the performance metrics used may not reflect a player’s skill, but rather reflect the player’s own style of play.

Instead of using the performance, what if there was a way of predicting the player’s skill from a short space of time? This would allow us to select a difficulty for the player using developer knowledge about the game’s difficulty, and not requiring the player to have played for a long period of time. So the question becomes, how can we predict a player’s skill?
1.5 What does this research offer?

Some research has gone into predicting skill in other domains [11] and in more general games [12]. However, skill prediction in video games is limited [13] and has not been done in a video game where the player directly controls an in-game avatar.

This research therefore proposes using the in-game events, including the player’s input to the game, to predict the player’s skill in a short space of time. Possible applications of this are automatic difficulty selection, discussed above, augmenting multiplayer matchmaking techniques, selecting appropriate opponents for each player according to their predicted skill, or collecting more information about players after release.

The primary application that we choose to explore here is that of automatic difficulty selection. To that end, the main question of the research is, “can skill be predicted in such a way that a difficulty can be selected for the player?” This is approached using seven hypotheses through the research, presented and answered in Chapter 3 and Chapter 4. They are listed here for convenience:

Hypothesis 1 A player’s skill in a first-person shooter can be predicted within a single game from their keyboard and mouse input better than using the performance measurement for that game.

Hypothesis 2 A player’s affective state can be predicted from their input to a first-person shooter better than a majority baseline (guessing the majority class).

Hypothesis 3 A player’s skill can be predicted in a single-player game after their first 30 s of gameplay significantly better than random guessing.

Hypothesis 4 The average difficulty and progress per second for games with a difficulty chosen using $s_E$ are at least as similar than those that use difficulty chosen using $s_C$.

Hypothesis 5 The experienced difficulty of games with a difficulty chosen with $s_E$ are at not significantly different to those that use $s_C$.

Hypothesis 6 Games with a difficulty chosen using $s_E$ are at least as fun than those with a difficulty chosen using $s_C$. 
Hypothesis 7: Players assigned a difficulty using $s_E$ experience as much flow as those using their self-assessment, $s_C$.

In testing these hypotheses, we present a number of contributions highlighted here:

- Two substantial data sets of game events in both a multiplayer and single-player first-person shooter [14]
- An analysis of skill in the context of first-person shooters
- An analysis of possible features with respect to skill
- A novel method for objectively measuring difficulty in single-player games
- Predictive models of skill for both games, capable of ranking players within 30 s of playing
- A demonstration of Automatic Difficulty Selection (ADS) applied to a comprehensive, commercial game engine.

1.6 How is this research laid out?

In the next chapter we present the reader with a detailed account of the current state of the research domain and the information required to understand the rest of this thesis. This includes a summary of the literature surrounding this research and the techniques we used.

This is then followed by the first of two research chapters, Chapter 3, that explores prediction of skill in a multiplayer game, Red Eclipse [15]. This chapter contains an in-depth study of skill measures within the game that examines the validity of different measures used, and studies the features extracted from the data set. Finally, the extracted features are used to predict the most suitable skill measures.

The work presented in Chapter 3 has been published as a conference paper [16] and accepted for publication in a journal [17]. The data set associated with this portion of the work can be found online [14].

The second of the two research chapters, Chapter 4, focusses on different methods of measuring skill in a single-player game, which inherently does not
rank players. A final study is then undertaken that, using another predictive model of skill, explores the ability to select an appropriate difficulty for a player, specifically in comparison to players’ self-assessments of their own skill.

In conclusion, we reflect on the results, discussing the strengths and weaknesses of the presented work, highlight the most significant findings and suggest some avenues for future research in Chapter 5.
Chapter 2

Background

The research we present needs to be presented in a clear context. The first step towards this is clearly defining the terms used in this research. Then the existing literature surrounding this field will be introduced, and particularly relevant research will be described and analysed, illustrating where foundations have been made and further work is yet to be done. Finally, we present the techniques used in the remainder of the research.

2.1 A short taxonomy of skill

Some of the common terms such as skill and difficulty have already been introduced. Here we define these more precisely in the context of video games and show how they relate to each other. The technical meaning of the terms may differ from that in everyday usage.

2.1.1 The skill, performance and difficulty triangle

We define the first term, and the most important in our research, skill, as the average level of performance over a set of games. This means that a value of skill only means something for a particular set of games and for a particular averaging technique. Comparing one skill measurement against another from a different measure does not make sense. For example, comparing someone’s number of kills against another’s completion time is a fallacy. This term is generalisable, so using different measures of performance and different games still results in meaningful measurements of skill.
This definition is distinct from the term ability, which Parker and Fleishman define as: “Ability refers to a more general, stable trait of the individual which may facilitate performance in a variety of tasks. . . . The term skill is more specific: it is task oriented.” [18]. Our definition of skill does not take into account drift, where a player may learn over time, or even decrease in performance. It also assumes that skill is averaged over a reasonable length of time.

We use the terms measure and measurement distinctly such that a measure is a system for producing a measurement, and a measurement is an instance of a measure. ‘Metres’, the measure, might produce a measurement of 10 m. In this way, we distinguish the abstracted skill measure, e.g. the score, from a player’s individual skill measurement.

The next term, performance, is a value assigned to a particular task after it has been completed. This value, or measurement, is defined by a criterion, or measure, where different measures may yield different performance measurements for the same task (e.g. the player with the highest score may have taken the most damage), and the choice of measure used affects the rankings of players.

The connection between skill and performance is illustrated in Figure 2.1 and is similar to the connection Chomsky draws between competence and performance in languages [19]. Highly competent people (with high skill measurements), can still perform badly, and likewise, it is still possible for a novice player to perform well on occasion.

Bergersen et al. defined a similar connection between terms while evaluating employee skill in programming [20]. While the terminology is similar, they do, however, use another term, ‘expertise’. In their research, expertise is still domain
2.1. A SHORT TAXONOMY OF SKILL

and task-specific, unlike ability, but at a higher level than skill, which is defined relative to smaller, representative tasks. In our research, we do not make this distinction, but assume that skill is defined by the task, regardless of its size.

**Challenge** is the difference between a player’s skill and the difficulty of the game. A highly skilled player will find less challenge from an easy game than a novice player, who may struggle (and therefore experience a higher level of challenge). Likewise, an easy difficulty may illicit the same level of challenge from a novice player as a hard difficulty from an expert. This term is not used formally in this research, and so is not defined strictly. A possible interpretation of challenge could be the expected performance for a given player on a game with a particular difficulty.

We could not find a formal definition of difficulty in the context of games. We therefore define it in relation to performance, as we did skill. **Difficulty** is the average performance for a piece of content over all values of skill. This allows us to describe difficulty in terms of the performance measures and players. Likewise, given a player and piece of content, we can work out the expected performance. This definition is explored further in Chapter 4 in the context of a single-player game.

In summary, skill is an attribute of the player, and the difficulty an attribute of the content. These two determine the challenge, which in turn determines the player’s expected performance for the content. The interplay between these concepts has been visualised in Figure 2.2. In this figure, the unknown property is skill, for both the developer and, in most cases, the player [10]. This highlights the importance of skill capture techniques.
2.1.2 Skill capture

In this research we define any method used to obtain a measurement of skill as **skill capture**, and is the umbrella term for any method used to obtain a representation of skill for a player. **Skill measures** are the most common example of these, obtained by averaging a player’s performance measurements over time. The averaging method used may vary depending on how the different games are chosen to be weighted for the player’s gaming history. TrueSkill, for instance, discussed later, places greater emphasis on recent games.

**Skill prediction**, on the other hand, is the method of predicting a skill measure using less data than would be available with the skill measure. It is done when one does not have enough data to create a reliable measurement of skill for a player.

The final term, **skill assessment**, is a subjective evaluation of someone’s skill. This may be using experts, a subject’s peers or self-assessment. A measure is commonly produced at the end of the process, allowing players to be compared or classified. Common examples include observations or, in other domains, critics of restaurants or films. The connections between these three skill capture terms have been presented in Figure 2.1, along with their connection to the previously defined terms.

2.1.3 Bloom’s taxonomy

Bloom *et al.* developed a taxonomy in educational psychology that defined educational objectives, separating them into three domains: 1) cognitive, 2) affective and 3) psychomotor [21]. The first of these, the cognitive domain, is about cognitive learning and concerns things like memory and reasoning. The second, the affective domain, refers to emotional growth and includes someone’s ability to sympathise with others. The final domain, the psychomotor, concerns physical skills, like the ability to play an instrument.

Each of these domains defines further categories that describe how a student learns within that domain. These terms are arranged in a hierarchical format, such that the lower categories must be mastered before progressing to the next. These taxonomies were designed to aid teachers and educators in teaching their students, and as such, concern skill acquisition.
2.1. A SHORT TAXONOMY OF SKILL

Although these are taxonomies of educational psychology, there is some overlap with skill acquisition in video games. This research does not consider skill acquisition and learning, but the categorisation of the domains are of interest, particularly the cognitive and psychomotor domains, which can be directly related to video games. These categorisations allow us to further understand the aspects of skill and how they may be measured separately.

2.1.4 Conroy’s model of skill

Bloom’s taxonomy breaks down skill into three digestible concepts. However, it was developed for a different domain, and as such, does not encompass all concepts of skill found within a video game. Conroy attempts to further break down skill into five main categories [22]:

- Mechanical dexterity
- Threat detection
- Tactical thinking
- Multi-tasking
- Prior knowledge

Conroy designed this model in order to better model AI’s threat detection within first-person shooters [23], and originally applied it to skill ceilings [22]. It can, however, be applied to other areas of skill, allowing us to separate an individual’s skill attributes into separate, distinct categories.

While Bloom split skill into cognitive and psychomotor domains [21], Conroy further splits the cognitive into distinct sections. Although this model has not been peer reviewed, the model provides a stepping stone for analysis of skill in video games and allows us to sufficiently categorise the different ways we are measuring skill.

The first term, ‘mechanical dexterity’, concerns the player’s ability to control the in-game character through the mechanical inputs, and is analogous to Bloom’s psychomotor categorisation. Differences of this may be seen across game platforms, where a player accustomed to a gamepad may see a drop in skill when moving to a mouse and keyboard, even if the game is otherwise the same.
The second term, ‘threat detection’, concerns the ability to assess threats to the player or AI. In a chess engine, this is analogous to the evaluation function that determines how strong a particular board configuration is for a player. Before using this model, we would change this term to ‘situational awareness’. Although the first term suits implementations of AI, the second is more abstract, and can be applied to scenarios that do not include enemies, such as recognising power-ups in a game.

‘Tactical thinking’ is the player’s ability to think strategically and use the prior knowledge and situational awareness to their advantage. Another take on this is how far the player can see ahead and how many different possibilities they can assess.

Conroy includes the term ‘multi-tasking’, which concerns how many of the different aspects of skill the player can perform at once. Common taxonomies of the psychomotor domain, however, include automation [24], where the student becomes adept enough that the task can be performed unconsciously. It may be, therefore, that this term belongs to a taxonomy of skill acquisition, rather than one of static skill.

The final term, ‘prior knowledge’, is how much knowledge the player has of the game. This may be, for instance, their knowledge of which keys perform which actions, or how certain units behave in a real-time strategy game.

### 2.1.5 Player Modelling: accounting for different kinds of people

The dawn of video games overlaps the emergence of machine learning, and so it is apt that the two gave birth to a new scientific field: the study of computational intelligence in games (CIG). This field studies how machine learning can be used to enable games to learn and adapt to the player, creating a more immersive and more enjoyable environment. CIG has been applied to artificial agents [25], content generation [26] and player modelling [27], which have all started to see some representation in the industry such as in Left 4 Dead’s AI director [28, 29], Far Cry 2’s dynamic trees [30] and SteamSaga’s use of the Five-Factor Model [31, 32].

Not all players are the same. Some players have different levels of skill, some have different favourite colours. Some players may also prefer FPS games, while
others prefer strategy-based games. Even within each genre, players can have
different preferences for different game modes or different weapons, or simply
have different ways of playing the same game.

Because players have different preferences, it can be difficult to match a game
appropriately so that the most people get the most enjoyment from it. One field
of study that serves to explore this is player modelling. Player modelling attempts
to learn the individual preferences of players such that the game can be tailored
to their style of playing or their particular likes and dislikes.

In the case of skill capture, the purpose is to model the player’s skill. In
other examples of player modelling, machine learning models are used to predict
a player’s preference within a game. In our own research, it is used to predict
their skill.

2.2 Dynamic Difficulty Adjustment (DDA)

In order to combat the differences in player skill and avoid players having to choose
the right difficulty, developers introduced DDA. One of the earliest examples
appeared in the 1986 game *Zanac* [33], using a system dubbed ‘automatic level
of difficulty control’ [34]. This approach also allowed for games to adapt to
the different learning rates of players. If a player’s skill increased dramatically
through the lifetime of one game, the game would, in theory, accommodate for
that. Here we introduce some examples of DDA and discuss their merits.

2.2.1 Commercial examples

DDA has been present in video games for several years now, implemented in ways
ranging from subtle to blatant, and it has worked with varying degrees of success.

The 2004 game *Half-Life 2* [6] employed a basic DDA system that was more
or less invisible. Whenever a player opened a supply crate, the contents would
depend on the player’s current state. A player on low health, for instance, would
be more likely to find a large health kit. Enemies would also occasionally drop
small health vials when the player was low on health. This simple and subtle
mechanic does not ‘reward’ the player for performing badly, it simply curbs the
risk of death slightly.

The *Left 4 Dead* series, by the same developer, is an example of more extensive
DDA. It employs an ‘AI Director’ that controls when enemies spawn and where
ammo and health will appear [29]. It is designed to respond to the players’
progress in a way that the tension of the game ebbs and flows, much like a film
director or orchestral conductor would.

The DDA in Bethesda’s Oblivion [35] was also extensive, but somewhat more
controversial. During the game, a player’s ‘level’ increases over time as the player
gains experience, and reflects the in-game skills of the player’s character. In order
to keep some areas imposing and less accessible, enemies would level up with the
player. This was done to the extent that higher level players would completely
stop seeing easier enemies and commonly come across over-powered monsters.
The most dramatic problems that this caused were related to quests which relied
on specific (low-level) creatures. In addition, players never felt rewarded for
levelling up.

The popular racing series Mario Kart [36] features power-ups that grant the
players with particular abilities such as invulnerability or weapons. These may
change depending on the player’s rank in the game. The game also provides
some ‘rubber banding’, where the highest ranked players are slowed down and
the drivers near the back are granted extra speed as if connected by an invisible
rubber band. Designed to enable players of different skill compete, this can cause
issues for players of similar skill, where being in first-place during the race is
undesirable.

Although the Call of Duty series has not been reported to use DDA, it does
employ a trivial method of difficulty recommendation [37]. Players are tasked to
complete an obstacle course and are recommended a difficulty based on the time
taken to complete it. Players may repeat the obstacle course or select a different
difficulty.

One final game to mention is God Hand [38], a game that integrated dynamic
difficulty into its very gameplay. Players were presented a bar that reflected
the DDA system, showing how well they were doing and how hard the enemies
were in response to this. By making the system completely transparent for the
player, players would not feel like they had to ‘cheat’ the system; rather, they
were expected to.

The games presented here offer a variety of DDA systems. Some, such as
Half-Life 2, are very trivial, while others, such as Oblivion, are more involved
with gameplay. Other than God Hand, which directly presents the player with
the underlying system, the systems that were less obvious to the player were those
that generally received the greater praise.

### 2.2.2 DDA in the literature

Although we do not directly employ DDA in our experiments, the task of assigning appropriate difficulties is closely related. Zook and Riedl put both processes under the term *challenge tailoring*, where appropriate content is assigned to a player [39]. The major difference between DDA and the method used in our experiments, difficulty selection, is that the former is an online process, while difficulty selection is performed statically. Similarly to [40], discussed later, the authors use the success rate of the player to determine their skill, but focus solely on the procedural memory of the player rather than their dexterity. The major contribution of this paper is an exploration of temporal mechanics in a game, which accounts for changes in player skill over time.

A common feature highlighted in this paper is that of a desired performance curve. In this way, the level of challenge experienced by the player varies over the game, keeping the player interest and adding points of interest such as high intensity events, like those present in *Left 4 Dead*. In order to match this curve, researchers used game data in order to train an adaptive model [41]. In this way, an adaptive model could be trained off-line and used for online DDA. The research demonstrates an effective way of continuously adapting difficulty through incremental changes to a game, but does not demonstrate the effectiveness on player satisfaction.

An alternative method to adapting the difficulty, other than a direct change in the NPC stats is changing their behaviour directly [42]. In order to accomplish this, researchers created opponents for a driving game that were defined by a series of behaviours. Example behaviours were the ability to reverse, a method for wasting time (avoiding achieving a particular goal) and turning through tight angles. By varying these behaviours, the agents were able to appropriately match their opponents abilities. While this research was not performed with humans, it demonstrates a possibility of natural difficulty matching. One shortcoming of the research, however, is that the agents must already be sufficiently good to provide the natural handicaps presented in the paper.
2.2.3 Shortcomings

The concept of DDA was developed in order for games to be set at the correct difficulty for the individual player without the developer having to know anything about the player in advance. However, in order to do this, the game assumes that a player’s performance is representative of their skill. This causes the DDA system to reward the player for doing badly: a poor performance will cause the game to get easier. Likewise, when the player does well, they are punished with a harder game. This can lead to players either gaming the system, intentionally playing bad to make enemies easier, feeling frustrated when they should be satisfied with their progress, or even feeling cheated when they start performing poorly.

2.3 Skill capture in the literature

While they have their shortcomings, DDA systems have the distinct advantage that the player’s skill does not need to be known in advance. If we do know the player’s skill, however, the difficulty system can be designed appropriately. Here we discuss different methods of capturing a player’s skill.

2.3.1 Measuring skill in video games

All competitive games have a winner and a loser, resulting in a crude performance measure with two states (three including draws). This win/loss measure can be generalized to a ranking system, where each player receives an ordinal number between 1 and $N$, the number of players, and draws are indicated by equal numbers. The winner is the player with a rank of 1 at the end of the game. This measure is considered the ‘gold standard’ of performance measures in that it directly reflects the winner of each game and fully encompasses the task assigned.

In video games, these ranks are commonly determined by another performance measure which varies depending on the game mode. Traditional ‘deathmatch’ modes, for instance, rank players by the number of kills while ‘capture the flag’ modes use the number of flags captured by each team. This can also be extended to other game modes that are objective-based. ‘Last man standing’, in which the last player alive wins, can be described by the number of deaths of each player. These measures are referred to as objective-based measures in this research, because they describe the objective assigned to the player.
Outside of these measures, players may additionally use their own measures for comparing players. *StarCraft* players, for instance, use a player’s average *actions-per-minute*, while *Counter-Strike* players use *accuracy* and *kill-to-death ratio*. However, these measures do not necessarily reflect the results of the game. As such, a player with a high accuracy may not win often. These are therefore referred to as community measures.

In a competitive game, the game mode defines the task, and each player understands what they have to do in order to accomplish it. As such, rankings are the only measures that truly reflect the player’s performance. On the other hand, rankings can be relatively undescrptive, usually producing a very small set of states. For a small set of games, objective-based measures can better distinguish between different players of similar rank. Indeed, objective-based measures have been used to improve existing skill models [43].

A common problem with measures is ‘inflation’, where players change their gameplay to manipulate their performance (and consequently their skill measurement), contrary to how the developers intended them to play. A combination of measures is therefore used to create a model of skill and encourage desired behaviour [44]. *World of Tanks* constructs such a model using weightings of different measures and a series of mathematical operations to produce a single skill measure [45].

These methods are great at measuring skill in a multiplayer game, where players compete against each other to produce rankings. In a single-player game, however, players are required to complete a game that consists of numerous tasks which may vary in style. As such, there is only one measure that can be used to reflect their performance: whether they completed the game or not.

That a player completed a game is not inherently descriptive. It may mean that the game was too easy for the player, or that they managed to find an exploit. Similarly, failing to complete it may reflect that the game was too hard as much as the player had no motivation to finish it. Moreover, it fails to describe which parts of the game the player struggled with.

Other performance measures may indicate the level of challenge the player experienced. Time taken is one example [46]; a longer time taken suggests the player struggled with parts of the game, while completing the game in a short amount of time implies the game was too easy. The amount of damage received may also reflect how well the player avoided getting shot, and thus say something
about their skill.

The main problem with any performance measure of a single-player game is that players have different player styles [47]. A player may choose to explore the environment and therefore take longer. Similarly, a rash player may choose to take damage from enemies in order to deal more damage. Neither of these players are, however, inherently bad as their performance measurements might suggest. As such, trying to measurement skill in a single-player game may cause difficulties.

2.3.2 TrueSkill: An example of a skill measure

TrueSkill averages performance using Bayesian updating [48], unlike some of the measures previously mentioned. The model is based on the Elo rating [49], which represents a belief in a player’s skill given a set of game results. This belief can be reduced to a single skill measure by combining the model’s parameters. The TrueSkill model uses game ranks as the performance measure of choice, and can therefore cope with multiple teams of varying player sizes.

Each player has two unitless values associated with them, \(\mu\) and \(\sigma\), which represents the skill the model believes the player has. The first value, \(\mu\), is the estimate of the player’s skill, and the second, \(\sigma\), the confidence the model has in that estimate. Higher values of \(\sigma\) mean the model is more unsure about the estimate assigned. Together, these values represent a normal distribution of skill.

At the start, each player is assigned equal values of \(\mu\) and \(\sigma\), where \(\sigma\) is relatively high to indicate the model currently knows nothing about the player. After each game, the player’s rank is used to update the values of \(\mu\) and \(\sigma\).

When two players compete, the two normal distributions can be combined to indicate the probability of a draw (the prior). After the game, the result (the likelihood), can be used to update the model’s belief in both players. If a player, Alice \((\mu_a, \sigma_a)\), beats Bob \((\mu_b, \sigma_b)\), \(\mu_a\) would increase, \(\mu_b\) would decrease, and both values of \(\sigma\) would decrease according to the following formulas:
2.3. SKILL CAPTURE IN THE LITERATURE

\[ \mu_{\text{winner}} \leftarrow \mu_{\text{winner}} + \frac{\sigma^2_{\text{winner}}}{c} \cdot V, \quad (2.1) \]

\[ \mu_{\text{loser}} \leftarrow \mu_{\text{loser}} - \frac{\sigma^2_{\text{loser}}}{c} \cdot V, \quad (2.2) \]

\[ \sigma^2_{\text{winner}} \leftarrow \sigma^2_{\text{winner}} \cdot \left(1 - \frac{\sigma^2_{\text{winner}}}{c^2} \cdot W\right), \quad (2.3) \]

\[ \sigma^2_{\text{loser}} \leftarrow \sigma^2_{\text{loser}} \cdot \left(1 - \frac{\sigma^2_{\text{loser}}}{c^2} \cdot W\right), \quad (2.4) \]

where

\[ c^2 = 2\beta^2 + \sigma^2_{\text{winner}} + \sigma^2_{\text{loser}}, \quad (2.5) \]

\[ V = v \left( \frac{\mu_{\text{winner}} - \mu_{\text{loser}}}{c}, \frac{\varepsilon}{c} \right), \quad (2.6) \]

\[ W = w \left( \frac{\mu_{\text{winner}} - \mu_{\text{loser}}}{c}, \frac{\varepsilon}{c} \right). \quad (2.7) \]

The functions, \( v \) and \( w \), dictate the update for \( \mu \) and \( \sigma \) respectively. The variable \( \varepsilon \) represents the probability of a draw and \( \beta^2 \) represents the player’s performance variance. If the players’ performance values are expected to vary more, the values will update slower. These values both require some previous knowledge about the game for sensible initialisation.

In order to produce a skill measurement from this model, \( \mu \) and \( \sigma \) are usually combined to produce a continuous value that can be used to rank players. Typically in TrueSkill, a pessimistic measure is used that gives lower values when the model is particularly unsure, specifically \( \mu - 3\sigma \). In other words, there is a 99.9% chance that the player’s actual skill measurement is greater than this estimate.

Because TrueSkill uses player rankings, it is extensible to any game with a league. When there are many players in each game, the model is also able to learn each player’s skill within a few games. Although reportedly close to the information-theoretic limit [48], when there are fewer players or teams, it becomes harder to converge\(^1\). Another criticism of TrueSkill is that measures cannot be compared across different leagues [51]. If players of one league are significantly more skilled than those of another, TrueSkill values will not reflect this, because players have only ever played against others in the same league.

---

\(^1\)Two teams of four players takes approximately 46 games to converge [50].
The bot-based TrueSkill estimate

TrueSkill is traditionally designed for multiplayer leagues where players compete directly against each other, and so their model parameters will interact with each other. Even if two players have never directly competed, they will have competed with similar players, and as such, their TrueSkill values will still be relative.

Unfortunately, participants in our experiments never competed directly against each other, but against artificial agents (bots). We therefore applied a slight adaptation to the TrueSkill algorithm, described here.

For each game, several opponents (bots) were selected randomly from a predefined range, \( b \), e.g. 40–50. So in a single game, there may be bots with a difficulty of 42, 43, 48 and 49. Each bot range was assigned an initial \( \mu_b \) and \( \sigma_b \); in other words, we assumed every bot in the range had the same \( \mu_b \) and \( \sigma_b \) values.

In order to update the values, a game was selected at random, and all bots assigned \( \mu_b \) and \( \sigma_b \) according to the difficulty range \( b \). At the end of the game, the bots’ ranks were used to update individual \( \mu_b \) and \( \sigma_b \) values, as in standard TrueSkill. These posterior values were then averaged to get an updated \( \mu_b \) and \( \sigma_b \) for the bot range. This was repeated until all games had been seen.

With estimated TrueSkill values for each bot range, we could assign these to bots as static place holders and then calculate player TrueSkill values, \( \mu_p \) and \( \sigma_p \) as normal. These resulting values would all then be relative to the bot difficulties.

2.3.3 Skill prediction in the literature

Skill prediction techniques try to predict these measures automatically within a short space of time. If this can be done accurately, players do not need to wait as long to experience the right level of challenge. Skill is not, however, restricted to the domain of video games, and some research in skill prediction has already been explored in other domains.

In end-game chess, where a limited number of moves are available, computer programs can consult databases to determine the optimal move in any position. Although limited to the very end of the game, comparing moves to the database can return a performance measure for each move [12].

Regan et al. extend this principle to entire games of chess [52]. By using the assumption that computers can play better than humans, the player’s move is compared with those of a computer to produce a prediction of the player’s
performance. Bayesian averaging is done over several moves to produce a skill prediction. This is a method of prediction simply because the optimal moves are not known.

Skill prediction has also been performed outside the scope of games in domains such as Human Computer Interaction (HCI) [11, 53]. These works use several features of the user’s mouse input to predict a user’s skill for a specific task, e.g. selecting a menu item. The features extracted were largely based on mouse movement. While FPS games require complex input from both the mouse and keyboard, the useful features they extracted have been carried across to our own research.

The other major difference between their research and ours is their focus on the predefined task. The user is given specific instructions to follow and repeat that the user can learn quickly between repetitions. In games, this would be equivalent to asking the player to move to a designated target or fire at a target. However, the authors also construct generalised models capable of predicting a user’s ‘system skill’, their general ability at using the system. This is more analogous to our own tasks, where players are given a single goal over the game and no further instructions.

Predicting skill is also important when selecting potential employees. Hiring the wrong employees can cause a company to lose a lot of money. As such, the company needs to know how skilled the applicant will be in the future. Take, for example, an entry-level job which requires no existing knowledge, companies cannot judge applicants based on their current knowledge, but have to predict how well they will gain the skills required to benefit the company. Schmidt and Hunter look at a variety of performance measures and skill assessments in order to work out which are the best predictors of future job skill [54]. Each of these are analysed and discussed in the context of both future profit and difficulty and cost in running the test. This research does not employ any machine learning techniques but it does demonstrate the analysis of performance measures, albeit in a different domain. As discussed in this article, skill assessment, such as unstructured interviews, can be subjective and expensive.

In round-robin tournaments, each player competes against every other player, which allows players to be ranked by their average performance. Ultimately, this method takes into account the interactions between all players, and can therefore create an exhaustive ranking of skill. However, it does not scale well. When there
are many participants, it is no longer feasible for everyone to compete against everyone else. Samothrakis et al. use the round-robin rankings as a ground truth and explore different methods of averaging the players’ performances using two Elo-based systems [44]. The competition examined used an asymmetrical score-based game, *Ms. Pac-Man* [55], where the protagonist, Ms. Pac-Man attempted to get a high score, while the enemy AI, the ghosts, tried to keep the score low. This meant that the scores had to be converted to win/loss measurements before being used in the Elo-based systems.

This research is of particular interest because the game used in the competition is typically a single-player game, but was presented in a multiplayer context, thereby allowing skill to be objectively measured. Moreover, the authors show that the ranking can be predicted within a quarter of the total games typically required. However, the purpose of the research required a particularly accurate representation of the ranking, as it was used to predict winners in a competitive environment. This level of accuracy is not as critical in our own research, where the predicted values are used for assigning difficulty, and prediction speed is instead the priority.

There are a variety of techniques for collecting data in a video game. A popular mode of data collection is through monitoring the player’s physiological responses to the game [56, 57]. This is particularly useful for measuring the affective response of a player without having to interrupt gameplay or rely on fallible responses [56]. This has been applied to predicting skill in a fighting game [40]. Within fighting games, players typically input combinations of button presses to perform special actions. If the timing of the combination is correct, the action is performed, else nothing happens. The researchers used this ‘success rate’ as a skill measure, using the player’s physiological data to predict this. While a novel technique that provides a foundation for further research, there was a small number of participants involved in the experiment and little analysis between the different types of player that took part. There is also a disadvantage with using physiological data, in that it can be intrusive, potentially breaking the players’ immersion, and is not viable to collect within typical gaming environments.

Alternatively, data can also be collected from high-level game events, such as the number of times players perform particular actions. This data can be collected automatically by the game and has been shown to be useful for other methods of prediction [58]. Mahlmann et al. use this type of data to attempt to predict
completion time in _Tomb Raider: Underworld_ [46]. However, the focus of the paper was primarily on predicting other aspects of the player, and the results of the skill prediction were inconclusive. In this work, the authors used the relative absolute error (RAE), introduced in Section 2.4.6, and concluded that the model may not be useful for adapting the game in real-time.

The most closely related research to ours was conducted using the real-time strategy (RTS) game _StarCraft II_ [13, 59]. _StarCraft II_ groups players into leagues that roughly represent skill groups. Players compete against other players within their own league, and when they start consistently winning or losing games, they go up or down to the next league respectively. Tetske _et al._ use these leagues to predict a player’s skill using the ‘actions’ they perform in the game. These actions consisted of the interactions between the player and the interface. This research shows that skill can be predicted within minutes of starting a game, but relies on leagues as measures, which only imply skill.

In summary, there has been a scattering of works in different domains that have attempted skill prediction. There is even less research in video games that seek to accomplish this. The closest work to our own research was done in an RTS [13], predicting game leagues from player actions. We highlight three main aspects currently missing from the literature: 1) a thorough analysis of skill measures used for prediction, 2) skill prediction in a single-player game, and 3) an application of skill prediction.

### 2.3.4 Skill assessment: The Game Expertise Questionnaire

There are few standard surveys that have been tried and tested in video game research and most of these apply the study of player emotions or interactions in video games [60, 61]. There are fewer that apply to measuring a player’s skill in a video game. There are some questions that regularly appear in different literature, such as the number of hours played per week [62, 63]. To the authors’ knowledge, however, there is no research extensively studying the effectiveness of these questions and how they relate to player modelling.

There does exist a single questionnaire that has been reported to be reused over multiple experiments: the Game Expertise Questionnaire (GEQ) [64]. This questionnaire is designed to capture various aspects of a player’s skill by asking
Figure 2.3: A sample of the Game Expertise Questionnaire for the Logic/Puzzle genre. Three questions are asked and the player is asked to circle any games played. The full questionnaire can be found in Appendix A.

A mix of objective and subjective questions about the player’s experience within games. In addition to asking the participant’s gender and the number of years they have been playing video games, the questionnaire asks three questions for each of the six genres:

1. Your **expertise** level (1-7) for your most extensively played game in that category

2. Your **current hours per week** on average (for the past 6 months)

3. If you’ve **ever** played the game type more than 5 hours a week (for example, in high school).

The participant is then asked to circle any extensively played games of that genre, or write any in. The layout of the questions has been shown in Figure 2.3, which shows a portion of the questionnaire. The full questionnaire is attached in Appendix A. A final question asks players to list other games for which they consider themselves an expert.

This questionnaire provided a solid foundation for assessing player skills in video games. As such, it was adapted as required for our single-player experiments. The changes that were made before its use have been described in Chapter 4.

### 2.3.5 Existing data sets

One of the contributions of this research is the data sets collected during the experiments. Although the field is relatively new in academia, there already exist some data sets in video games. Here we highlight some that are relevant to our own research in order to provide some context for our own data sets.
The first of the data sets to discuss is The Platformer Experience Dataset [62], designed around a public Super Mario Bros clone, Infinite Mario Bros. The experiment was designed to model player experience using facial expressions and body postures, and as such, includes video of participants during play. In addition, features of each generated level and data regarding player in-game behaviour were recorded. Players also answered a brief demographic questionnaire and an experience questionnaire after each pair of games. In total, there were 58 participants and 380 games, a medium-sized data set, roughly equivalent to the Red Eclipse data set presented in Chapter 3. In general, the data is quite extensive and includes player behaviour data, potentially useful for predicting player skill at games. Unfortunately, there are no specific measures of skill recorded for each participant.

The second data set was constructed from StarCraft play logs known as ‘replays’. Replays contain enough game data to reconstruct a complete game. These replays were used to collate a list of events in the game that included information about the game state and the player’s behaviour, such as their orders and attacks [65]. This data set has been applied to learn tactical models for decision-making and ‘army composition’, where units engaged in attacks are grouped into the same army. This data set is much more substantial than any of the other data sets presented in this thesis, containing over 7,000 game sessions collated from professional game leagues and tournaments. However, while extensive and thorough in terms of player behaviour, the data set does not contain any information about player experience or the demographics of the players.

In a PC-based FPS, players are traditionally required to use the keyboard and mouse to control their in-game character. This dual combination of the two devices provides a more complex base to found our research on. Moreover, the players are required to move around in a 3-dimensional environment with very little constraint. This is in contrast to predator-prey games, where players are typically confined to tracks, platform games, where players can only move in a two-dimensional plane, and strategy games, where players cannot directly control the in-game characters at all.

Moreover, all our data sets include, in addition to the in-game performance measures, some evaluation of player skill external to the game in the form of questionnaire answers. The data sets also include, in addition to the high-level events such as player actions, lower-level player input-based events like key presses.
While there already exist some data sets in video games, they are not directly relevant for the prediction of skill. Our data sets are also, to the authors’ knowledge, the first data sets that use first-person shooters.

2.4 Machine learning techniques

The techniques used in this thesis are described in this section. Each is, in turn, introduced and described and its advantages and disadvantages are highlighted in the context of this work and how it might affect results. As with any domain, there are some terms that are central to machine learning that first need to be defined.

In this research we denote single example with \( x \) and its associated label with \( y \). The example, \( x \), is a vector of \( D \) features, each denoted with \( x_i \). A function (also known as a model) that maps an example to a prediction, \( \hat{y} = f(x) \), where \( \hat{y} \in \{k_1, \ldots k_M\} \), is a classifier. If the predicted value is continuous, \( \hat{y} \in \mathbb{R} \), the function is an example of regression.

In the context of a set of \( N \) examples, \( X \), each example is denoted by \( X_i \). A model is trained on a training set, \( X_{\text{train}} \in \{X_1, \ldots, X_t\} \). To show how well the model generalises and to avoid overfitting, a separate testing set is used, \( X_{\text{test}} \in \{X_{t+1}, \ldots, X_N\} \). To improve the assessment of the model’s generalisability, the data is split into a number of equal portions, known as folds. Each is used in turn as the testing set and the model trained on the remaining data. This is known as cross-validation.

Further details about the field of machine learning can be read in [66].

2.4.1 Random Forests

Several different models have been used successfully in previous player modelling research. Support vector machines (SVMs) have found success in skill prediction [13], while artificial neural networks have been used to predict player preferences [58]. While predicting behaviour, Mahlmann et al. report the success of 7 different classification algorithms [46], two of which are variants of a decision tree.

Random forests [67] are an ensemble method that build on decision trees. Their major advantage is that they are able to generalise well, even with a large number of features with unknown properties. They can also be used as a ‘grey box’ with little knowledge of their internal mechanics which can be exposed if
required, revealing the importance of different features while training. These properties are taken advantage of in our research. Finally, random forests can be employed for either classification or regression depending on how the separate decision trees are aggregated. This is particularly important when one needs to accommodate the different shapes and sizes of skill measures.

An individual decision tree, the building block of random forests, is constructed by recursively splitting the data into separate classes, at each point splitting on a single feature that produces the most information gain. If followed to completion, each leaf node in a tree will only contain examples of a single class. A testing example can be classified by following the branches of the tree, comparing the given feature at each level against the criteria defined when the tree was trained.

One problem that can occur with decision trees is that they can overfit to the training data, adapting to the random noise in the training data, rather than the signal to be modelled. Instead, separate trees are constructed using different subsets of data, each overfitting in slightly different ways. As a group, any errors caused by overfitting are averaged out, and generalisability is maintained. These subsets are drawn from the data set with replacement and only consider a randomly selected subset of the available features [67].

Once a random forest has been constructed, a testing example can be passed to each tree individually. Every tree will have a slightly different opinion on how the example should be classified, so a majority vote is taken. Due to the slightly different viewpoints of each of the trees, each tree will likely be quite good at classifying some examples, and so the errors of some trees are made up for by others. This is what causes the higher performance over simple decision trees. The way votes are combined can be varied to change the classification method. Averaging the votes, for instance, can produce a regression value.

This model has two settings that can be adjusted during training, \texttt{ntree} and \texttt{mtry}. The first, \texttt{ntree}, dictates how many trees to use and the latter how many features are sampled from whenever a tree is split.

### 2.4.2 Support Vector Machines (SVMs)

In a binary classification problem, where linear classification is used, a common problem can be to misplace the classification. Logistic regression, for example, may successfully classify a set of training examples like line B in Figure 2.4 but
Figure 2.4: Two possible decision boundaries for the same data. Both classify the existing data in the same way. Line A, however, has more room for error given any new points.

easily misclassify some of the testing examples. Line A, on the other hand, generalises better because it is further away from both groups.

Support vector machines attempt to maximise this margin between points and the decision boundary, thus increasing the model’s generalisation performance [68]. The closest points to the decision boundary are known as the ‘support vectors’; if they are removed, the decision boundary would change. However, this means that once the model is trained, the whole model can be represented by these single examples. The other points have outlived their usefulness and can be omitted while testing. This also makes support vectors quick when classifying. Unlike random forests, for example, which need to consult hundreds of trees, each which needs to be traversed, an SVM only has to compare the point against the relevant support vectors.

Another property of classifiers like logistic regression is that they are stochastic, reliant on the initialisation parameters of the model. SVMs, on the other hand, are reliable. Given a set of examples, there will be one optimal decision boundary with maximum margins.

Sometimes, however, the data may be noisier than we would like. In these cases, the SVM can be trained with an ‘error parameter’ that describes how lenient the SVM is to misclassified points in the training set. This is, in other words, a trade-off between increasing the margins and reducing the number of
misclassifications.

So far, we have talked about SVMs as linear classifiers. However, by taking advantage of ‘kernels’, the data can be transformed in such a way that the resulting data is linearly separable. This technique, known as ‘the kernel trick’, is visualised in Figure 2.5. The example shown here is a ‘polynomial kernel’ (the feature, $x$, is taken to a power) and is still quite regular. A radial basis function (RBF) is an even more flexible kernel that can create more obscure boundaries and is the kernel of choice in this research. The RBF can be further customised with two parameters: $\gamma$ and $C$. The $\gamma$ parameter dictates the radius of the RBF where a larger value of $\gamma$ dictates a smaller radius. $C$, on the other hand, determines the complexity of the boundary, trading simplicity for the ability to correctly classify more training examples, and potentially overfit.

The models so far have been examples of support vector classification, where an individual class is obtained from the model. In this research, however, support vector regression (SVR) is used instead to produce a continuous value for the given example. This was particularly useful where a skill measure needed to be predicted for players. In regression, examples are instead assigned continuous values that the model attempts to predict. The SVR model attempts to find values for the training data that are a close to the original as possible, as demonstrated in Figure 2.6.
CHAPTER 2. BACKGROUND

2.4.3 Linear Discriminant Analysis (LDA)

LDA is a supervised technique used for feature reduction [69], reducing the dimensionality while preserving as much information as possible, or, less commonly, as a linear classifier. Other methods for reducing dimensionality, such as PCA [70], look for the dimensions of the data with the most variance. LDA, on the other hand, attempts to maximise the difference between groups of different classes, rather than the data as a whole. This difference is visualised in Figure 2.7, in which line A might be the most discriminative dimension for the data as a whole, but does not capture the difference in classes like line B.

The main reason to using a dimensionality reduction technique such as LDA is to remove uninformative features. Some predictive models such as SVMs can perform badly when fed too many features with little information. Moreover, the resulting features, as a combination of the original features, may contain information that is more informative to the models (i.e. makes separating instances of different classes easier). LDA in particular has been used previously in player modelling [27]. However, LDA makes the assumptions that the data is normally distributed and the features are statistically independent. If the data is structured in a more complex manner, these structures may be distorted and lost before it is passed to a classifier.

In LDA, each example, $\mathbf{x}$, has $D$ features and a label, $y$. For simplicity, we start by assuming that there are only two classes. We are looking for the ideal
2.4. MACHINE LEARNING TECHNIQUES

Figure 2.7: Projecting the data onto vector A would keep the majority of the data’s variance. However, transforming the data onto B allows better differentiation between the two classes.

A dimension that maximises the amount of discrimination between two classes, as in Figure 2.7. In other words, we are looking for some vector, \( \mathbf{w} \), that can transform our example, \( \mathbf{x} \), onto this new axis to produce a single scalar value:

\[
\tilde{x} = \mathbf{w}^T \mathbf{x}.
\]  

(2.8)

The average position for a class, \( y_i \), is given as a vector of averages over each of \( \mathbf{x} \)'s dimensions:

\[
\mu_i = \frac{1}{N_i} \sum_{\mathbf{x} \in y_i} \mathbf{x},
\]

(2.9)

so the average position of each class in the transformed feature space (on our ideal line), is given by \( \tilde{\mu}_i \):
Figure 2.8: The dimension $x_1$ has a greater between-class scatter, but greater within-class scatter, meaning $x_2$ is more suitable.

\[
\tilde{\mu}_i = \frac{1}{N_i} \sum_{x \in y_i} \tilde{x}, \\
= \frac{1}{N_i} \sum_{x \in y_i} w^T x, \\
= w^T \mu_i. \tag{2.10}
\]

The principle behind LDA involves trying to keep classes as far apart from each other as possible so that it is easier to distinguish between them. Therefore, we need to start by separating their midpoints, $\tilde{\mu}$. Fisher suggests a function called the between-class scatter \[69\] that can be maximised:

\[
\tilde{s}_B = |\tilde{\mu}_1 - \tilde{\mu}_2|^2. \tag{2.11}
\]

However, simply keeping the centres of the classes apart may have issues if the classes are spread out. In Figure 2.8, we see that although the between-class scatter is larger on $x_1$, the values on $x_2$ do not overlap. Therefore, we need to measure the amount of variance within each class:
\[ s_i^2 = \sum_{\tilde{x} \in y_i} (\tilde{x} - \tilde{\mu}_i)^2. \] (2.12)

The sum of these values for each class is known as the within-class scatter \( s_W = s_1^2 + s_2^2 \). The Fisher criterion is the ratio of the between-class to the within-class scatter, and is the value that we are looking to maximise:

\[ J(w) = \frac{s_B}{s_W}. \] (2.13)

In order to maximise the separation, these values are defined in terms of the transformed feature space, \( \tilde{x} \) and \( \tilde{\mu}_i \). However, in order to find the transformation matrix, \( w \), these need to be defined in terms of \( x \) and \( w \):

\[ s_B = (\tilde{\mu}_1 - \tilde{\mu}_2)^2, \]
\[ = (w^T \mu_1 - w^T \mu_2)^2, \]
\[ = w^T (\mu_1 - \mu_2) (\mu_1 - \mu_2)^T w, \]
\[ = w^T S_B w, \] (2.14)

\[ s_i^2 = \sum_{\tilde{x} \in y_i} (\tilde{x} - \tilde{\mu}_i)^2, \]
\[ = \sum_{x \in y_i} (w^T x - w^T \mu_i)^2, \]
\[ = \sum_{x \in y_i} w^T (x - \mu_i) (x - \mu_i)^T w, \]
\[ = w^T S_i w, \] (2.15)

and

\[ s_W = s_1^2 + s_2^2, \]
\[ = w^T S_1 w + w^T S_2 w, \]
\[ = w^T S_W w, \] (2.16)
where
\[
S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T, \quad (2.17)
\]
\[
S_i = \sum_{x \in y_i} (x - \mu_i)(x - \mu_i)^T, \quad (2.18)
\]
\[
S_W = S_1 + S_2. \quad (2.19)
\]

The final step is to find the vector, \( \mathbf{w}^* \), that maximises the Fisher criterion, \( J(\mathbf{w}) \):
\[
\mathbf{w}^* = \arg \max \left[ \mathbf{w}^T S_B \mathbf{w} \right], \quad (2.20)
\]
\[
= S_W^{-1}(\mu_1 - \mu_2). \quad (2.20)
\]

This generalises well to a multi-class problem with \( C \) classes. In this case, LDA will reduce the data to \( C - 1 \) dimensions. We therefore need \( C - 1 \) projections, \( w_i \), which become the projection matrix \( \mathbf{W} \). Our projected space then becomes:
\[
\tilde{\mathbf{x}} = \mathbf{W}^T \mathbf{x}. \quad (2.21)
\]

The within-class scatter defined above is gracefully adapted to multiple classes by summing over each class. The between-class scatter, on the hand, incorporates the mean of all values, \( \mu \). The two matrices, \( S_W \) and \( S_B \) become:
\[
S_W = \sum_{i=1}^{C} S_i, \quad (2.22)
\]
\[
S_B = \sum_{i=1}^{C} N_i(\mu_i - \mu)(\mu_i - \mu)^T, \quad (2.23)
\]

where
\[
\mu = \frac{1}{N} \sum x. \quad (2.24)
\]

Using these values, we try to find the optimal matrix, \( \mathbf{W}^* \), that maximises \( J(\mathbf{W}) \):
2.4. MACHINE LEARNING TECHNIQUES

\begin{equation}
J(W) = \frac{|W^T S_B W|}{|W^T S_W W|}.
\end{equation} \tag{2.25}

It can be shown that the component vectors, \( w_i \), of this optimal matrix, \( W^* \), is the solution to:

\begin{align*}
(S_B - S_W \lambda_i) w_i &= 0, \\
S_W^{-1} S_B w_i &= w \lambda_i. \tag{2.26}
\end{align*}

In other words, \( W^* \) is composed of the eigenvectors\(^2\) of \( S_W^{-1} S_B \). Moreover, the eigenvalues for each of these eigenvectors indicate the scale of separability for each class, allowing the dimensions of \( W^* \) to be sorted by significance, and the top \( k \)-most features selected for dimensionality reduction, where \( k \ll D \).

Once complete, the user has a matrix, \( W \), which is \( D \times k \) in size. Any new samples can be simply transformed by multiplying the vector (\( D \times 1 \) in size) by \( W \), thereby reducing it down to \( k \times 1 \).

2.4.4 Feature selection (wrapper methods)

Sometimes a data set may have some completely redundant features that don’t contribute to the prediction or even hinder it by adding noise. Techniques like random forests can cope with this because different subsets of features are used at different points. For other techniques, such as an SVM, this can hurt performance.

There exists, for every model, some ideal combination of features that maximises the testing performance because of the way those features work together to provide the most information about the class labels. In an ideal world, we would test a model on every possible combination of features to find this subset. Unfortunately, for large numbers of features this would take much too long; testing models can be slow. Instead, one method is to progressively remove features from the model in turn, seeing how well the model performs without each feature \[71\].

This method of feature selection has the advantage that the success of the feature is directly related to the model, rather than testing features independently.

\(^2\)An eigenvector of \( A \) is one that, when applied to \( A \), scales \( A \). In other words, \( Av = kv \).
of the model. This means that features that may be relevant to other models can be ignored where necessary and does not rely on any particular regularity of the feature or its relevance to the labels to be notable. However, it also means that wrapper methods may be prone to overfitting if there are not sufficient examples. These methods can be very slow to train, particular for a large number of features. This method does have the added side effect that features can be ordered by their relevance by using the performance of the model after their removal. These performance values are useful in a critical analysis of the features.

2.4.5 Feature extraction

While some researchers have provided the foundation for skill prediction in video games [53, 13], one of the goals of this work is to understand better how a player’s behaviour and input to the game relates to their skill. In order to accomplish this, we require a variety of methods for extracting features from the player’s data. Here we present the methods that we employed to extract the appropriate features.

Measuring complexity

It is plausible that there is some difference in complexity between the mouse and keyboard presses of players of different skills. In other words, it may be that a player of high skill has significantly more complex input to the game than their novice counterparts, or vice versa. This feature of the data is called its entropy; how unpredictable it is. The more unpredictable the data, the higher its entropy.

We use a variety of methods for measuring the data’s complexity. The first of these involved data compression techniques, measuring how much the original string could be compressed. The larger the ratio between the compressed string and the original, the less complex it was, and therefore the lower the entropy. In order to test this, we used a variety of methods of measuring complexity.

The first method, Lempel-Ziv-Welch (LZW) [72], linearly constructs a dictionary from recurring sequences of characters. At each point, it consumes as much of the sequence as it can and replaces it with an existing entry in the dictionary. The algorithm is relatively simple to implement and is widely used:

1. Initialise a dictionary with single-character strings from the input.

2. Find the next longest string, $W$, that exists in the dictionary.
3. Replace \( W \) with the dictionary index.

4. Add \((W + \text{next character})\) to the dictionary.

5. Go to Step 2.

The second algorithm, designed by Huffman [73], takes advantage of probability distributions of the data to construct a Huffman tree. Common characters are given smaller codes so that they take up less space. To encode a string, characters are replaced using the codes in the tree. If the population of the characters is known, Huffman encoding is close to the theoretical minimum.

Each symbol in the alphabet, \( a \), has a probability or weight of its occurrence, \( w \). The simplest implementation uses a priority queue, where priority is given to nodes with lower weights:

1. Create a node for each symbol, \( a_i \), with weight \( w_i \) and add to the priority queue.

2. While there are two or more nodes on the queue:
   (a) Fetch the two nodes, \( n_1 \) and \( n_2 \), with the highest priority.
   (b) Create a node, \( n \), with weight \( = w_1 + w_2 \) and children \( n_1 \) and \( n_2 \).
   (c) Add node \( n \) to the queue.

3. The last node is the root node.

Once the tree is completed, the symbols can be replaced by the appropriate codes that represent the symbol’s position in the tree. The more frequent the symbol, the shorter the code.

In addition to using the compression techniques above, we also used the Shannon entropy [74], which measures the amount of information in a given sequence:

\[
H(X) = - \sum_i P(x_i) \log P(x_i). \tag{2.27}
\]

This equation sums the probability of all possible states of \( x, x_i \).

The final complexity measure used employed a discrete Fourier transform [75]. A signal is usually composed of several frequencies of different amplitudes.
Figure 2.9: A single signal composed of four sine waves of different frequencies and amplitudes.

Figure 2.10: The signal in Figure 2.9 broken down into its composite parts.

superimposed over each other as seen in Figure 2.9. A Fourier transform is a method for converting that signal back into its component parts, and thereby working out the relative strengths of the frequencies, as in Figure 2.10. For the analysis of continuous input like mouse input, this may illuminate any regularities that occur in the data.
2.4.6 Evaluating skill models

Skill measurements always have a total ordering. This means that classification models, which don’t assume any connection between classes, may not be appropriate, even in the case of discrete measures. This is particularly apparent when the models are evaluated: if \( x = 1 \), the predicted values \( \hat{x} = 2 \) and \( \hat{x} = 5 \) are punished equally, even though the first is closer to the original value.

We therefore look to regression models, which produce continuous predictions, regardless of whether the measure is continuous or discrete. This means that, in our example above, where \( x = 1 \), the model is able to output \( \hat{x} = 1.6 \).

Regression models are typically evaluated using the proportion of explained variance (\( R^2 \)) or the relative absolute error (RAE) [46]. Both of these measure how close the predicted values are to the original values. However, this means that models that offset or scale the results are punished, even if the predicted values observe the same ordering as the original values.

In our research, we are comparing values that are ultimately used for ranking players. For example, two skill measures that produce different measurements for four players (1, 2, 3, 4 and 30, 35, 90, 200) are equivalent so long as the rankings of the players are the same. Previous research has used the mean-squared error [44]. While this is useful for comparing performance for the same data, it does not scale for larger or smaller sets of players. We therefore use Spearman’s rank correlation coefficient (Spearman’s \( \rho \)) to evaluate the performance of our skill models. This also means that any two measures can be compared. Spearman’s rank is defined as the Pearson’s correlation coefficient of the ranking of the variables. Each point, \( x_i \), is converted to a rank, \( \tilde{x}_i \), where \( 1 \leq \tilde{x} \leq N \). The formal definition is:

\[
\rho = 1 - \frac{6 \sum (\tilde{x} - \tilde{y})^2}{N(N^2 - 1)}. \tag{2.28}
\]

The output of Spearman’s rank lies between -1 and 1, where 1 indicates perfect agreement between ranks, 0 indicates no relationship between the two measures, and -1 indicates the skill measures are the inverse equivalent of each other. As such, we generally report the magnitude of Spearman’s rank, \(|\rho|\).

In addition, it also necessary to compare the skill of different groups of players and determine whether they are statistically different. In this situation, the skill measure used may be non-parametric and the measurements are independent As
such, the Mann–Whitney $U$ test is suitable [76], a test which, unlike the more common t-test, is suitable for non-normally distributed data and has comparable efficiency to t-tests even on normally distributed data [77]. In our research, we use this test to determine whether one group of players is statistically more skilled than another given different significance levels, $\alpha$.

For two sets, $S_1$ and $S_2$, with sizes $N_1$ and $N_2$, first assign a rank to each value in each set, as with Spearman’s $\rho$, so $x_i$ becomes $\tilde{x}_i$. However, if there is a tie between values, take the average position of all values in a tie. 1, 2, 2, 4 would therefore become 1, 2.5, 2.5, 4. The $U$-values can then be calculated with:

\[
R_i = \sum_{x_i \in S_i} \tilde{x}_i, \tag{2.29}
\]
\[
U_i = N_1N_2 + \frac{N_i(N_i + 1)}{2} - R_i. \tag{2.30}
\]

For sets with a small number of observations, the smaller of the two $U$-values can be compared to a lookup table. For larger samples, $U$ will be approximately normally distributed, so can be converted to a $z$-value:

\[
z = \frac{U - \mu}{\sigma}, \tag{2.31}\]

where

\[
\mu = \frac{N_1N_2}{2}, \tag{2.32}\]
\[
\sigma = \sqrt{\frac{N_1N_2(N_1 + N_2 + 1)}{12}}. \tag{2.33}\]

The Mann–Whitney $U$ test is used to determine whether two samples were taken from different populations. This translates to showing whether or not one sample has a higher skill than another. However, sometimes it is necessary to compare the variance of two samples, for instance, to show that one sample is more unpredictable than another. In this case, a Brown–Forsythe test is appropriate [78], because it does not assume the samples are normally distributed.

In order to compare the variances, Brown–Forsythe first transforms the data, $x_{i,j}$ so that it is relative to the median of each group, $\tilde{x}_j$: 
2.4. MACHINE LEARNING TECHNIQUES

\[ z_{i,j} = |x_{i,j} - \bar{x}_j|. \] (2.34)

The following then works out the ratio between the between the amount of variance within each group and that across the groups:

\[ F = \frac{(N - K) \sum_j N_j (\bar{z}_j - \bar{z})^2}{(K - 1) \sum_j \sum_i (z_{i,j} - \bar{z}_j)^2}, \] (2.35)

where

\[ \bar{z}_j = \frac{1}{N_j} \sum_i x_{i,j}, \] (2.36)

and

\[ \bar{z} = \frac{1}{N} \sum_j \sum_i x_{i,j}, \] (2.37)

and where \( N \) is the total number of observations, \( N_j \) the number of observations for group \( j \), and \( K \) the number of groups. This test is used in Section 4.5.3 to determine the consistency of our models.
Chapter 3

Skill Prediction in a Multiplayer Game

Skill is a particularly important factor in multiplayer games, where a player must face off against other players of different skills. Matching these skills is particularly important in both multiplayer and a single-player contexts and several solutions have already been suggested that measure skill over a longer period of time [48, 79]. However, accurately predicting player skills would provide insight quicker, allowing games to be more appropriately matched sooner.

Some of the work presented in Section 2.3.3 began to explore this field, predicting skill in a variety of genres. The most complex example was demonstrated in StarCraft II [13], predicting skill within the first few minutes of gameplay. However, the input to real-time strategy games like StarCraft II is much like using a typical user interface, with cursor control and keyboard shortcuts. Many other genres, including FPSs, give the player direct control over a character. Moreover, there has been no extensive research into how to appropriately measure skill in the domain of games.

In order to demonstrate the feasibility of skill prediction in an FPS, a competitive, multiplayer game was chosen. A clear ranking can be obtained from competitive games and, as such, players can be assigned meaningful measurements. Many single-player games, however, as discussed further in the next chapter, do not define the task so clearly\(^1\) and skill is therefore harder to measure. In addition, multiplayer sessions are typically shorter than single-player campaigns.

\(^1\)Many arcade games, such as Mario or Pacman, provide the player with a score that accumulates during the game. These scores can therefore be compared with other players.
This made them ideal to use as a foundation for skill capture.

In this chapter we describe the experiment we undertook in order to demonstrate that a player’s skill could be predicted within a single game. Included in this section is an analysis of the skill measures used and features extracted.

## 3.1 Problem statement

Previous research has shown that skill could be predicted in a real-time strategy game using player actions [13]. However, players interact with a first-person shooter in a completely different way; controlling an in-game avatar directly. Research in HCI has also shown that player input to a computer was related to their skill [53]. The goal was therefore to determine whether a player’s skill could be predicted using their input to the game.

A secondary goal was accomplishing this in a reasonable length of time. Skill measures are only meaningful after several games [50], equivalent to an hour or more of gameplay. In the meantime, players must compete against poorly matched opponents. A prediction within the first game was therefore considered reasonable. Given these goals, we present the following hypothesis:

**Hypothesis 1** A player’s skill in a first-person shooter can be predicted within a single game from their keyboard and mouse input better than using the performance measurement for that game.

In order to test this hypothesis, we first needed to understand the skill and **performance measures** used. There are commonly several methods for measuring performance in a video game. Some of these measures are more descriptive than others. Rank, for example, which is a positive natural number, is only as descriptive as the number of players in each game. Some measures may be more ‘valid’ than others, in that they better describe the players’ accomplishment of the defined objective.

**Skill measures**, as an average of the above performance measures, will also have an inherent validity. In order to construct the most valid models of player skill, we should use the most valid skill measures. In this research, we make the assumption that a mean average has the most validity, although Bayesian averaging, e.g. using TrueSkill, may be equally or more valid.
To our knowledge, there is no extensive comparison of skill measures in a
game and how each accurately represents the player’s actual skill. This chapter
therefore includes a comparison of skill measures used in first-person shooters.

Some features of user input have been used in previous research [53]. However,
these are limited to mouse input. As such, we extract and analyse a series of
features that describe the player’s input to the game.

Previous research has shown player emotions can be predicted from game data
[58] and physiological data [56]. Given a new data set that extensively recorded
player input to the game using two forms of input (the keyboard and mouse),
we wanted to know whether it was feasible to predict the player’s affective state
from this. The hypothesis was as follows:

**Hypothesis 2** *A player’s affective state can be predicted from their input to a
first-person shooter better than a majority baseline (guessing the majority class).*

In conclusion, we therefore briefly present the results of trying to predict three
affective states; fun, frustration and challenge.

### 3.2 Experimental setup

The test-bed for this experiment was the cross-platform first-person shooter,
*Red Eclipse*[15]. Built on the engine for *Cube 2: Sauerbraten*[80], the game’s
distinctive features include real-time level editing and an open-source licence\(^2\).
Along with the game’s support for customisation of settings, this facilitated the
construction of the test-bed. In addition, *Red Eclipse* is representative of modern
first-person shooters, in that it includes many common game mechanics from the
genre.

*Red Eclipse* also includes some slightly different game mechanics to other first-
person shooters. While health that regenerates over a few seconds is becoming
more common in first-person shooters [37], *Red Eclipse* uses a health bar that
slowly regenerates over a longer period of time. For this reason, *Red Eclipse*
doesn’t contain any health kits, unlike some of its contemporaries [81]. Similarly,
*Red Eclipse* does not provide ammo packs. Instead, each weapon has unlimited
ammo. Finally, *Red Eclipse* also includes a ‘parkour’ system that allows players
greater freedom in moving around their environment. In recent years, there have

\(^2\) *Red Eclipse* is covered under the ZLIB licence.
been a select number of first-person shooters that employ this mechanic such as *Brink* [82]. The key associated with this parkour system was pressed by many of the players in our experiment, but only two players used it consistently throughout their games. We therefore dismiss any effect this system has on predicting skill.

When playing a multiplayer game, a player can choose the type of game they play, the **game mode**, the simulated arena in which they want to play, the **map**, and, when simulated enemies known as **bots** are available, their difficulty. In addition to these settings, *Red Eclipse* also offers a host of customisable settings, such as weapon damage and the number of weapons that can be carried. In our experiment, most settings were kept on their default settings.

Three settings in particular were adjusted for the experiment. After preliminary trials, the game speed was decreased to 80% to be more accessible to new players. The time limit was set to 3 minutes. This was deemed long enough to be immersive, but short enough to mitigate memory effects [83] and avoid lengthy experiments and boredom [57]. Finally, the game mode was set to ‘deathmatch’, in which players compete to kill each other for the most points. This game mode is straightforward, therefore accessible, and meant players were not reliant on the skill of their team.

Two settings were then changed for each game: the map and the bot difficulty. Eight different maps were used in the experiment, which represented a range of environments. Some maps were larger than others, some more complex, and some had greater vertical variety. In addition, some maps were easier for the player, while others were easier for the bots. The basic specifications for each of the maps is given in Table 3.1.

A game’s difficulty is described by a minimum and maximum which define the range of possible difficulties assigned to bots in that game. Each bot in that game is randomly assigned an integer from that range, inclusive of the limits. This value defined the bot’s ‘skill’ for that game. In our experiment, six non-overlapping ranges were used (40–50, 50–60, 60–70, 80–90 and 90–100).

### 3.2.1 The Log File

*Red Eclipse* was instrumented so that each game created its own log file, populated with metadata and timestamped events. These log files, originally comma-separated have been published as **JSON** objects for flexibility [14]. The structure of the log files was based on similar research looking at data collection in an HCI
Table 3.1: Details for each map used in the experiment.

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Players</th>
<th>Environmental hazards</th>
<th>Median reported complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bath</td>
<td>Small</td>
<td>6</td>
<td>No</td>
<td>2</td>
</tr>
<tr>
<td>Deli</td>
<td>Large</td>
<td>8</td>
<td>No</td>
<td>3</td>
</tr>
<tr>
<td>Echo</td>
<td>Small</td>
<td>6</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>Error</td>
<td>Large</td>
<td>6</td>
<td>Yes</td>
<td>3</td>
</tr>
<tr>
<td>Foundation</td>
<td>Medium</td>
<td>6</td>
<td>No</td>
<td>2</td>
</tr>
<tr>
<td>Ghost</td>
<td>Medium</td>
<td>6</td>
<td>Yes</td>
<td>2</td>
</tr>
<tr>
<td>Ubik</td>
<td>Large</td>
<td>8</td>
<td>Yes</td>
<td>3</td>
</tr>
<tr>
<td>Wet</td>
<td>Medium</td>
<td>6</td>
<td>No</td>
<td>2</td>
</tr>
</tbody>
</table>

environment [84].

The log file’s metadata defines the game and its settings. The list of metadata has been presented in Table 3.2 along with a brief description. Although the bot difficulties in this table are listed as having a greater range, they were restricted to between 40 and 100 as difficulties lower than 40 were considered minimally different for the purposes of this experiment.

The events recorded in the log file fell into two classifications: input events and game events. Input events were fired whenever the player pressed a key, moved the mouse or pressed a mouse button. Keyboard and mouse button events both contained a key or button identifier, the final state of the button and the action the button caused in the game. Mouse motion events, on the other hand, recorded the x and y position of the mouse in pixels and were sampled roughly once every three milliseconds while the mouse was in motion.

The second category of events was limited to some higher-level interactions of the player with the game. Only interactions that involved the player were recorded in this experiment. In other words, one bot killing another was not recorded. The exception to this was the ‘score’ event, which was triggered when anyone gained or lost points. The game events have been detailed in Table 3.3.

3.2.2 Data collection

The experiment was run in-house, which enabled more control, giving both consistency and reliability to the data set. It also meant that settings could be tailored to keep the data set balanced throughout the experiment.
3.2. EXPERIMENTAL SETUP

Table 3.2: The metadata for each game.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Game ID</td>
<td>A unique identifier for the game.</td>
<td>127</td>
</tr>
<tr>
<td>Player ID</td>
<td>A unique identifier for the current player.</td>
<td>26</td>
</tr>
<tr>
<td>Client Number</td>
<td>The number assigned to the player by the game. Always 0 in this experiment.</td>
<td>0</td>
</tr>
<tr>
<td>Game Number</td>
<td>From the set of games played by one player, the position this game appears</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>(starting from 0).</td>
<td></td>
</tr>
<tr>
<td>Map Name</td>
<td>The name of the map that was selected for this game.</td>
<td>wet</td>
</tr>
<tr>
<td>Bot Min</td>
<td>Each bot’s difficulty is chosen randomly from between Bot Min and Bot Max.</td>
<td>60</td>
</tr>
<tr>
<td>Bot Max</td>
<td>Possible values range from 0 to 101.</td>
<td>70</td>
</tr>
<tr>
<td>Connect time</td>
<td>The time the user connected to the game (ms).</td>
<td>1</td>
</tr>
<tr>
<td>Disconnect time</td>
<td>The time the game ended (ms).</td>
<td>185010</td>
</tr>
<tr>
<td>Scoreboard</td>
<td>The final scoreboard for the game, including number of points and kills for</td>
<td>{0: {'points': 8,</td>
</tr>
<tr>
<td></td>
<td>each player (given by their client number).</td>
<td>'kills': 3} ...}</td>
</tr>
<tr>
<td>Date &amp; time</td>
<td>The date the game was played and the time it started.</td>
<td>2013-02-26, 14:40:54</td>
</tr>
</tbody>
</table>

Figure 3.1: The overall format of the experiment.

In this research we have attempted to use the term participant in the context of the experiment, and player in the context of the game. The terms are otherwise synonymous.

The structure of the experiment was similar to a previous study [57], and has been visualised in Figure 3.1. Participants started by completing a consent form and demographic questionnaire, which asked six questions about background, such as age, and playing experience.
Table 3.3: Game events recorded by *Red Eclipse*.

<table>
<thead>
<tr>
<th>Name</th>
<th>Triggered</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spawn</td>
<td>The player reappears in the game (e.g. after they were killed)</td>
<td>1</td>
</tr>
<tr>
<td>Death</td>
<td>The player is killed by the specified opponent (or themselves)</td>
<td>8</td>
</tr>
<tr>
<td>Kill</td>
<td>The player kills the specified opponent.</td>
<td>8</td>
</tr>
<tr>
<td>Assist</td>
<td>The player contributes to the death of an opponent.</td>
<td>2</td>
</tr>
<tr>
<td>Damage received</td>
<td>The player takes damage from the specified source.</td>
<td>6</td>
</tr>
<tr>
<td>Damage dealt</td>
<td>The player deals some damage to the specified opponent.</td>
<td>6</td>
</tr>
<tr>
<td>Health regeneration</td>
<td>The player receives a portion of health.</td>
<td>3</td>
</tr>
<tr>
<td>Points</td>
<td>Some points are awarded to the specified player.</td>
<td>3</td>
</tr>
<tr>
<td>Ammo used</td>
<td>The player fires the specified weapon.</td>
<td>4</td>
</tr>
</tbody>
</table>

Participants were then given a written tutorial and time to read it. The tutorial included a summary of general FPS mechanics and details about the specific mechanics of *Red Eclipse* such as the parkour system and the weapons available. Participants were not obliged to read it, however, but could ask questions through the experiment or refer to the tutorial again as they needed. Information was not otherwise volunteered by the researcher.

The rest of the experiment was separated into ‘sessions’, which consisted of two games and three questionnaires, as in Figure 3.1. The first of the two games was named ‘Game A’, the second ‘Game B’. After each, the participant filled in a Likert-based questionnaire about their experiment. At the end of the second questionnaire, they were asked to fill in the third questionnaire that compared the two games. These questionnaires are further described in the next subsection. The player was allowed to complete as many whole sessions as they wished.

All participants used the same keyboard and mouse, and a headset was provided to wear at their discretion. The researcher was present throughout the experiment to guide participants and answer any questions. On three occasions,
the researcher had to intervene to ensure participants completed the questionnaires for the previous games. For each of these games, there is roughly an 18 s gap of missing game data.

After an initial study, the experiment was rerun several months later in order to correct for imbalances in the data and increase the amount of data available. This means that some games were separated by a significant amount of time. 14 of the 45 participants only took part in the first study, there were 11 new participants in the second, and 20 participants took part in both.

### 3.2.3 Questionnaires

As shown in Figure 3.1, the player completed four different questionnaires during the experiment: a demographic questionnaire, two Likert-based questionnaires and a questionnaire that uses 4 Alternative Forced Choice (4-AFC) [85].

The demographic questionnaire was presented to participants before they started and gleaned information such as age, gender and, most notably, two self-reported measures of skill. Not having access to the Game Expertise Questionnaire at the time, the skill-based questions were designed to reflect the player's skill in the game. The first measure, how many hours the participant plays per week, is a common question in research [63, 86]. The second, the number of first-person shooters played \( f \), was conceived to discount the effect of other genres and account for the player's entire gaming experience, rather than playing habits. These questions were designed to be objective and avoid self-assessment, which players are notoriously poor at [10]. A third question asked whether the player had ever played *Red Eclipse* before. The questions were worded as follows:

- How many hours do you usually play video games in a week?
  - 0 - 2, 2 - 5, 5 - 10, or 10+

- How many first-person shooters have you played previously?
  - Never, 1 or 2, 2 - 5, 5 - 10, or 10+

- Have you ever played *Red Eclipse* before?
  - Y / N

From the group of 20 players that took part in both testing periods, 6 players gave different answers for the second question between the two experiments.
While most of these discrepancies were off by one category, one participant reported playing more (1 or 2 went up to 5 - 10), while another reported playing less (10+ down to 2 - 5). Any effects caused by differences in skill over this time period have been ignored. Instead, the participants’ first answers were used.

The three experienced-based questionnaires presented to the participant asked the same questions in slightly different ways. The first two used a Likert scale [87], asking players to give a numerical answer between 1 and 5 for each game. The Likert questionnaires were worded as follows:

- How much would you want to keep playing the game?
- How frustrating did you find the game?
- How challenging did you find the game?
- How lost did you feel while playing the map?

The final questionnaire used 4-AFC and asked the participant to compare the previous two games. Each of the above questions was reworded as follows:

- Which did you want to keep playing most?
- Which game was most frustrating?
- Which game was most challenging?
- Which map was the most complex?

In answer to this, the participant could choose ‘Game A’, ‘Game B’, ‘neither’ or ‘both equally’. There are advantages and disadvantages to both methods of measuring response, Likert and 4-AFC, which are discussed more thoroughly in [89]. Likert, for example, is often prone to order-effects, while 4-AFC is not as descriptive.
3.2. EXPERIMENTAL SETUP

Figure 3.2: The number of games played by each player. Games highlighted by the dashed box are those used in this research. Colors indicate which maps each player played.

3.2.4 Data Distribution

Over the two experiments, 45 participants played a total of 476 games. All participants played at least 4 games, most played at least 8, and one participant played 22 games. This has been visualised in Figure 3.2. This data set was published in its entirety [14], but in order to reduce bias, a subset of 430 games from 37 participants was used to generate the results presented in this chapter, highlighted in Figure 3.2.

Some effort went into making sure there was a balance of skills. After the preliminary study, the number of FPSs played ($f$) was found to be the better measure of self-reported skill, and was therefore used to validate the balance of data across players, maps played and bot difficulties. For each game, the map and bot difficulty was selected pseudorandomly, in order to ensure that each player and each group of $f$ played a roughly even number of games on each map and difficulty.

The maps played by each player have been overlaid on Figure 3.2. In addition, the map and bot difficulties for each skill group is shown in Figure 3.3 and 3.4 respectively for the entire data set. Although there is an overall imbalance of players according to $f$ (there are very few players in the ‘Never’ category), the distribution of the population is unknown. In other words, there may be relatively very few people that have never played a video game before. It may also be that
Figure 3.3: The number of times each map was played, overlaid by the number of times played by each group in $f$ (FPSs Played).

Figure 3.4: As in Figure 3.3, the number of games played on each difficulty, with additional grouping over $f$ (FPSs Played).

players are less likely to select certain answers due to biases or memory effects; the category ‘2 - 5’ is also smaller than others. However, for this experiment, the distribution was considered acceptable. In particular, the split between those that played less than 5 games and those that played more was roughly even.

In the first study, average performance levelled out after the 6th game. We therefore discarded any players who had played fewer than 8 games, and ignored any games played after the 16th in order to minimize bias. There were 430 games
3.3 Measuring skill

There are numerous ways of measuring skill in a video game. Some of these are more valid than others, in that they more accurately represent how skilled the player is at completing the given task. Other measures of skill, however, may better represent a player’s behaviour or preferences in the game. The number of shots fired, for instance, may represent how wasteful the player is more than how often they win.

Some skill measures can also distinguish between players better than others. This may depend on the number of possible measurements or the frequency of events. In Counter-Strike: Source [90], players can only die once per round. As such, players may regularly have the same number of kills. In contrast with Team Fortress 2 [81], where players are revived regularly, the measure is not very descriptive.

In this section, we consider the validity of several skill measures, defined here as the magnitude of correlation, Spearman’s rank ($|\rho|$), to a given measure. We also consider the descriptiveness of each of these, allowing us to select the measure that is the most descriptive of players, while remaining valid in the context of our task, a deathmatch. Moreover, further research can build upon these findings for other tasks or other genres. Each of the skill measures used has been summarised with their notation in Table 3.4 for convenience.

3.3.1 Rank

For any symmetrical game [44], where each player is given the same task to accomplish, rank is the performance measure that defines the relative success of each player in accomplishing this task. For the $i$th player, $r_i$ is equal to the number of players that performed the task better than them. In other words, $r = 1$ indicates the winner, $r = 2$, second place, and so on. Ties are indicated by $r_i = r_j$. For team-based games, this generally indicates the ranking of the teams rather than players. In our experiment, $r$ is the player’s position at the end of the game relative to the bots they played against.
Table 3.4: A summary of the skill measures in *Red Eclipse*.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player rank $\bar{r}$</td>
<td>Mean rank ($r$) over all of an individual’s games.</td>
</tr>
<tr>
<td>Player score $\bar{s}$</td>
<td>Mean score ($s$) over all of an individual’s games.</td>
</tr>
<tr>
<td>TrueSkill estimate $T$</td>
<td>A TrueSkill value produced using an estimation of initial TrueSkill values for bots. This is described further in Section 2.3.2.</td>
</tr>
<tr>
<td>FPSs played $f$</td>
<td>The number of FPSs the player reported they had played.</td>
</tr>
<tr>
<td>Hours played $h$</td>
<td>The number of hours the player reported they played per week.</td>
</tr>
<tr>
<td>Player KDR $\bar{k}$</td>
<td>Mean kill-to-death ratio ($k$) over all of an individual’s games.</td>
</tr>
<tr>
<td>Player accuracy $\bar{a}$</td>
<td>Mean accuracy ($a$) over all of an individual’s games.</td>
</tr>
<tr>
<td>Player deaths $\bar{d}$</td>
<td>Mean number of deaths ($d$) over all of an individual’s games.</td>
</tr>
</tbody>
</table>

Rank is the most valid performance measure by definition; it defines the task and success of each player. It was defined in Chapter 2 as the gold standard of performance measures for this reason. Rank is, unlike other measures we consider later, more invariant to content; a win on one map has the same $r$ value as a win on a different map. However, $r$ is defined by $P$, the number of players for that game. One consequence of this is that $r$ does not hold validity between games of different size $P$. Another consequence is that for games of small $P$, it is not very descriptive, as there are fewer possible values of $r$. Thus it becomes harder to distinguish between two players with similar performance.

As a performance measure, $r$ is not completely invariant to content. The performance values for each map is presented in Figure 3.5. Although there is
3.3. MEASURING SKILL

Figure 3.5: This Tukey box plot [1] presents the performance measure, rank \((r)\) and skill measure \((\bar{r})\) for every game, grouped by the game’s map. Here, a lower value indicates higher performance. On average, players performed slightly worse on the maps Foundation and Wet.

Figure 3.6: Rank \((r)\) and player rank \((\bar{r})\) presented in the same notation as Figure 3.5, grouped instead by difficulty. Harder difficulties (e.g. 90–100) led to much lower performance.

some variation seen for each map, using a Mann–Whitney U test, there were no statistical differences found between any two maps for a significance level of \(\alpha = 0.005\). The most significant difference was between the maps Echo and Foundation, with \(p = 0.0277\). There were, however, statistical differences between ranks on different difficulties, seen in Figure 3.6.
A skill measurement for each player is obtained by averaging their rank over all their games. This produces the continuous skill measure player rank, $\bar{r}$. This is compared to $r$ for map and bot difficulty in Figure 3.5 and Figure 3.6 respectively, demonstrating that there is no dependence on either content. This method of averaging, using the mean, works in this case because players experienced similar contents. Later in this section, however, $r$ is averaged using Bayesian updating (TrueSkill) instead.

### 3.3.2 Score

In order to work out the player’s rankings, games will often use some measure of performance. Racing games, for example, use time taken, while fighting games use health remaining. Traditionally, the goal of a deathmatch is to accrue the most amount of kills, and the game ends either when a particular number of kills is reached, or a pre-specified amount of time has passed. In a *Red Eclipse* deathmatch, players must instead accrue the most amount of points. Points are awarded for killing other players, with bonus points for other skilful behaviour such as killing multiple people within a short period of time. At the end of the game, players are ranked according to their score, $s$.

Because players are ranked by their score, $s$ still has validity. In other game modes or genres, it may not. *Team Fortress 2*, for instance, keeps a score for each player, although these values do not directly impact the results of the game. Indeed, any other arbitrary composition of a score measure would need to be tested for validity.

The main advantage of $s$ over $r$ is that $s$ has a much larger range of values, and is therefore more descriptive. A larger value of $s$, for instance, may imply an easier victory. However, Mann–Whitney $U$ tests ($\alpha = 0.005$) between the map-based distributions (Figure 3.7) show that the players achieved a higher score on *Echo* than 6 of the other maps, while *Foundation* had a statistically higher score than *Ubik* and *Wet*. Both of these maps were smaller than others and reported as less complex by players. Although not as pronounced, difficulty 90–100 has a lower average score than difficulties 40–50 and 60–70, as shown in Figure 3.8. These dependencies on content mean that score cannot be compared across content.

A skill measure was created from $s$ called player score, $\bar{s}$, by taking the mean of a player’s $s$ measurements over all their games, as with $\bar{r}$. Figure 3.9 visualises
3.3. MEASURING SKILL

Figure 3.7: As in Figure 3.5, but using performance and skill metrics score \((s)\) and player score \((\bar{s})\) respectively. Higher \(s\) indicates higher performance.

Figure 3.8: As in Figure 3.6, but with score \((s)\) and player score \((\bar{s})\). As with \(r\), \(s\) was lower for higher difficulties.

This, showing the performance measurements, \(s\), for each player, overlaid with their respective score measurements, \(\bar{s}\). It is visible from this figure that players with consistently higher scores are assigned a high skill measurement. This measure also accounts for outliers, highlighted in the figure, where the player achieves a high score not typical of their skill.

In Figure 3.10, the distribution of \(\bar{s}\) has been directly compared to \(\bar{r}\). It can clearly be seen that rank skews towards the more skilled players, i.e. ranks of 1.
CHAPTER 3. SKILL PREDICTION IN A MULTIPLAYER GAME

Figure 3.9: Score \( (s) \) for each player, ordered by the players’ mean scores \( (\bar{s}) \). Games highlighted in red indicate unexpectedly high values of \( s \).

Figure 3.10: Distribution of player ranks \( (\bar{r}) \) compared to the distribution of player scores \( (\bar{s}) \).

This is caused by the fact that, on average, bots were less skilled than players. In a fully multiplayer experiment, this distribution of \( \bar{r} \) should be normal. However, for our purposes, \( \bar{s} \) demonstrates greater differentiation between the two extremes: skilled and novice players.

We also determine that the number of games required before a reasonable measurement of player skill can be averaged. If \( \bar{s}_i \) is the mean score for a player over their first \( i \) games, we show the correlation between \( \bar{s}_i \) and \( \bar{s} \) in Figure 3.11.
3.3. MEASURING SKILL

Figure 3.11: The correlation, $|\rho|$ between $\bar{s}_i$ and final player score ($\bar{s}$), where $\bar{s}_i$ is calculated by averaging score ($s$) over the first $i$ games for each player.

Table 3.5: The score groups separated by player score ($\bar{s}$).

<table>
<thead>
<tr>
<th>$\bar{s}$</th>
<th>Name</th>
<th>Number of Players</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 14</td>
<td>Novice</td>
<td>9</td>
</tr>
<tr>
<td>14–22</td>
<td>Intermediate</td>
<td>10</td>
</tr>
<tr>
<td>22–27</td>
<td>Skilled</td>
<td>9</td>
</tr>
<tr>
<td>$\geq$ 27</td>
<td>Expert</td>
<td>9</td>
</tr>
</tbody>
</table>

Between $\bar{s}_5$ and $\bar{s}_7$ the correlation begins to converge. A number of players only played 8 games, hence the large increase in $|\rho|$ between $\bar{s}_7$ and $\bar{s}_8$. We therefore argue that the number of games used in these experiments is sufficient to provide a measure of player skill.

For some purposes, players need to be grouped by skill. We therefore separate players into four bins according to their $\bar{s}$ measurements. These groups, defined in Table 3.5 and visualised in Figure 3.12, were constructed such that there were an equal number of players in each group. A Mann–Whitney $U$ test confirms that the groups’ average performances ($s$) are statistically different from each other with a significance level of $\alpha = 0.005$. 
3.3.3 TrueSkill

In addition to a mean, we also averaged rank using the TrueSkill algorithm to produce a $T$ value for each player. The estimated TrueSkill value, $T$, and $\sigma$ for each score group has been presented in Figure 3.13 and Figure 3.14 respectively. Although the 50–60 $T$ value is slightly higher than that of 60–70, each has a $\sigma = 0.638$.

The $T$ values have been visualised against $\bar{r}$ and $\bar{s}$ in Figure 3.15. In addition, the correlation between all skill measures discussed in this chapter has been presented in Table 3.6. Given the high correlation of TrueSkill with $\bar{r}$, we argue that it is valid in the context of our task. A number of players in Figure 3.15 with high $T$ values ($T > 25$) have a greater spread on the $\bar{s}$ scale than $T$. We argue that $\bar{s}$ is therefore more descriptive because it better discriminates between skills.

3.3.4 Self-reported measures

Asking players about their gaming experience is common in related research [86]. It can serve to put research into context, and is very easy data to collect. We therefore explore two self-reported measures of skill, hours played ($h$) and FPSs played ($f$) in order to determine their validity.

Both of the measures explored here attempt to ask the player an objective
3.3. MEASURING SKILL

Figure 3.13: Average TrueSkill estimates ($T$) before each game for each score group. From bottom to top, each dotted line represents a bot difficulty; 40–50, 60–70, 50–60, 70–80, 80–90 and 90–100.

Figure 3.14: Convergence of the average TrueSkill $\sigma$ for each score group over time.

question; something that can be measured. This is in comparison to asking them directly for an assessment of their own skill which people are notoriously poor at [10].

The $s$ and $\bar{s}$ values for each of the groups in $h$ has been shown in Figure 3.16. The only statistical difference between neighbouring groups with a Mann–Whitney $U$ test is for $h = 0–2$ and $h = 2–5$ for $s$, with a significance level of $\alpha = 0.005$. There were, however, only three players in the group $h = 5–10$. As
such, few conclusive remarks can be made. There is a relatively low correlation between $h$ and previously examined skill measures such as $\bar{s}$ ($|\rho| = 0.501$). This implies there is a low level of validity in $h$.

The second measure, $f$, separates players into five categories. Rather than $h$, which only considers recent playing history, this measure attempts to account for the participant’s entire gaming history and exclude time spent playing other genres. A comparison between $f$ and $\bar{s}$ is shown in Figure 3.17. Although more closely
3.3. MEASURING SKILL

Figure 3.16: The distribution of score (s) and player score (\( \bar{s} \)) for each hours played (h) group.

Figure 3.17: The distribution of score (s) and player score (\( \bar{s} \)) for each FPSs played (f) group.

correlated (\(|\rho| = 0.770\)), only 2 participants fell under the group \( f = \) Never. Between neighbouring pairs, \( f = 2–5 \) was significantly less than \( f = 5–10 \) with a significance level of \( \alpha = 0.005 \). There were also some differences between non-neighbouring groups found, such as between \( f = 1 \) or 2 and \( f = 5–10 \). These differences were more notable than those between the groups of h, implying a greater level of validity.
3.3.5 Community measures

The gaming community will often use game statistics to evaluate and compare players. These are designed to give a better understanding of each player’s strengths and weaknesses, but are often specific to the game genre they are used for, such as a player’s actions-per-minute in StarCraft. Two common performance measures used in FPS games are kill-to-death ratio, $k$, often abbreviated KDR, and accuracy, $a$.

A player’s KDR is the number of kills they obtained per player. Given that kills help players achieve the objective and dying is undesirable, KDR is a sensible method for measuring how efficient each player is. Two players with the same skill measure could be differentiated by how efficiently they finished the task. Indeed, some games, such as the Counter-Strike series, use this measure as a tie-breaker. A skill measure has been calculated for this using a mean to create player KDR, $\bar{k}$. The correlation between $\bar{k}$ and $\bar{r}$ shows that there is a fairly reasonable level of validity between the two measures ($|\rho| = 0.8476$). As shown in Figure 3.18, the measure seems less able to differentiate between players of higher skill. These differences may be down to play style; some players prefer a riskier approach.

A player’s KDR, which uses the number of kills, directly influences the player’s score, and, therefore, their rank. The number of deaths, however, generally does not change the player’s score and only indirectly influences their rank (being killed increases the scores of other players). We therefore measured the number...
of deaths per game, $d$, and plot the average deaths per player, $\bar{d}$ against $\bar{r}$. It would be expected that novice players are more likely to die (there is some correlation with rank, $|\rho| = 0.3069$), but the validity of $\bar{d}$ is the lowest of the measures presented here. The number of times a player dies may therefore relate more to play style [27].

The final performance measure, a player’s accuracy, is calculated by dividing the number of times they hit the opponent by the total number of shots they fired. Similarly to KDR, this measure incorporates a sense of efficiency; more skilled players will waste less ammunition. Unlike the other measures discussed in this section, a player’s accuracy does not influence the player’s rank. The player could choose to continuously fire their weapon, even when no enemy is in sight, or only ever fire when a hit is guaranteed.

As with $k$, $a$ has been averaged over a player’s games to produce $\bar{a}$, player accuracy. The relationship between $\bar{a}$ and $\bar{r}$ is visualised in Figure 3.19. The correlation between the two measures, $|\rho| = 0.6811$, agrees with our initial assumption, that skilled players are more efficient. However, there is less difference seen in accuracy between the more highly skilled players. This may imply that accuracy is an ability more quickly mastered.
3.3.6 Summary

The validity of each of the skill measures discussed in this section is listed in Table 3.6. The two skill measures, \( \bar{s} \) and \( T \), have the most validity when compared to the player’s mean rank, \( \bar{r} \). Given that \( T \) was only an estimate of TrueSkill, and due to the extra descriptiveness present in \( \bar{s} \), we chose \( \bar{s} \) to use as the primary measure for this analysis.

Of the two self-reported measures, \( f \) was found to be somewhat descriptive of skill, but still too ambiguous to be used for predictive models. It may, however, be useful to use in other research, where a reasonable measure of a participant’s skill needs to be obtained without asking them to play many games. \( h \), on the other hand, showed less validity. Moreover, the differences between groups were not statistically noticeable. In other words, it did not capture a player’s skill particularly well.

The three measures that were not taken from task-oriented measures, \( \bar{k} \), \( \bar{d} \) and \( \bar{a} \), were generally correlated with skill. However, the average number of deaths, \( \bar{d} \) was the most interesting because it had the least validity. The correlation with player score was so low, \(|\rho| = 0.1156\), that one could infer that it better measures a player’s style of play. Each of these measures may have different validity for other types of FPS. In a game where ammunition is scarce, for instance, \( \bar{a} \) may better describe skill.

3.4 Feature extraction

In order to predict skill, a player’s game must first be converted to a series of features. While some features had been explored in previous work [53], other features needed to be invented. Using the methods described in Section 2.4, we extracted 174 global features from the keyboard and mouse events for each game. The features are analysed in this section in order to better understand player input and its relationship to skill. The complete list of features can be found in Appendix B for reference.

The features were grouped using three different schemes, summarised in Table 3.7. Each of these groups is designed to describe the input in a different way. In each scheme, features are assigned to a single category or left ungrouped if they do not belong to any category or belong in more than one.

The relationship of the features to skill is shown using the Pearson correlation
Table 3.7: Feature groups used within this research

<table>
<thead>
<tr>
<th>Group name</th>
<th>Description</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>From keyboard events</td>
<td>83</td>
</tr>
<tr>
<td>Mouse</td>
<td>From mouse movement events</td>
<td>66</td>
</tr>
<tr>
<td>Clicks</td>
<td>From mouse clicks</td>
<td>14</td>
</tr>
<tr>
<td>Ungrouped</td>
<td>-</td>
<td>11</td>
</tr>
<tr>
<td>Event Frequency</td>
<td>Frequency of events over the game</td>
<td>31</td>
</tr>
<tr>
<td>Complexity</td>
<td>Complexity of input</td>
<td>75</td>
</tr>
<tr>
<td>Kinetics</td>
<td>Describing how the player or mouse moves</td>
<td>19</td>
</tr>
<tr>
<td>Ungrouped</td>
<td>-</td>
<td>49</td>
</tr>
<tr>
<td>Context-Free</td>
<td>No prior knowledge of game required</td>
<td>78</td>
</tr>
<tr>
<td>Dependent</td>
<td>Some knowledge of game semantics needed</td>
<td>96</td>
</tr>
</tbody>
</table>

coefficient (Pearson’s $r$) with respect to player score, $\bar{s}$, chosen as a major index of skill in this experiment. Strong correlation is defined as $r = 0.6$, slightly greater than suggested in previous work [91]. The features with the greatest correlation have been summarized in Table 3.8.
Table 3.8: The top ten features ranked according to their correlation to player score ($\bar{s}$). The entire list of features can be found in Appendix B.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Hardware</th>
<th>Type</th>
<th>Context</th>
<th>Pearson’s $r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>lz-bintostr-keybin</td>
<td>The LZW complexity of the keys’ binary string.</td>
<td>Keyboard</td>
<td>Complexity</td>
<td>Dependent</td>
<td>0.7994</td>
</tr>
<tr>
<td>mean-numkeys</td>
<td>The average number of keys pressed at one time.</td>
<td>Keyboard</td>
<td>Free</td>
<td></td>
<td>0.7896</td>
</tr>
<tr>
<td>length-movement</td>
<td>The total length of the player’s movement path.</td>
<td>Keyboard</td>
<td>Kinetics</td>
<td>Dependent</td>
<td>0.7804</td>
</tr>
<tr>
<td>multikeys/2</td>
<td>The total time for which at least two keys are pressed.</td>
<td>Keyboard</td>
<td>Free</td>
<td></td>
<td>0.7802</td>
</tr>
<tr>
<td>sum-magnitude-displacement-clickBursts-movement</td>
<td>The player’s movement during clicking bursts (periods where the player performs lots of clicking).</td>
<td>Clicks</td>
<td>Dependent</td>
<td>0.7738</td>
<td></td>
</tr>
<tr>
<td>keytime-requiredkeys</td>
<td>The total time the basic set of keys are pressed.</td>
<td>Keyboard</td>
<td>Event Frequency</td>
<td>Dependent</td>
<td>0.7663</td>
</tr>
<tr>
<td>length-position</td>
<td>The length of the player’s position path.</td>
<td>Keyboard</td>
<td>Kinetics</td>
<td>Dependent</td>
<td>0.7662</td>
</tr>
<tr>
<td>lz-intostr-keybin</td>
<td>The LZW complexity of the keys’ ASCII string.</td>
<td>Keyboard</td>
<td>Complexity</td>
<td>Dependent</td>
<td>0.7579</td>
</tr>
<tr>
<td>zlib-bintostr-keybin</td>
<td>The zlib complexity of the keys’ binary string.</td>
<td>Keyboard</td>
<td>Complexity</td>
<td>Dependent</td>
<td>0.7578</td>
</tr>
<tr>
<td>num-keyevents</td>
<td>The number of key press events.</td>
<td>Keyboard</td>
<td>Event Frequency</td>
<td>Free</td>
<td>0.7573</td>
</tr>
</tbody>
</table>
3.4.1 **Hardware**: keyboard, mouse movement and clicks

The *hardware* grouping separates features according to the input device that generated the events. One of the first obstacles to playing a game is learning how to use the input devices. As such, it is likely to contribute to skill. The question for this categorisation was therefore to determine which input device best described skill.

The features extracted from the *keyboard* events concerned the complexity of the input or the frequency with which they were pressed. Some of these features were extracted from the movement keys that allow the player to move around, for example the amount of time the player spent strafing left and right. A number of mouse movement events have already been used in related HCI research [53], and these formed the basis for the *mouse* features. Mouse *clicks* had the fewest features because they were the simplest in nature and have been used less in the literature. Some of the features extracted using this form of input were based on ‘click bursts’; groups of mouse click events that were close together. The features that are ungrouped in *hardware* were based on an estimate of the player’s current position and used both the mouse and keyboard. These were therefore ignored in this grouping.

The overall correlation of the features in each group has been presented in Figure 3.20. Although the *keyboard* group contains the most features, it was also one of the more interesting groups, as most features were correlated in some way. The *mouse* group, on the other hand, correlated significantly less with skill overall. This contrasts previous work in HCI, in which mouse features played a key role [53]. A possible cause of this is that the mouse is constantly in motion in an FPS, therefore causing more noise. The objectives in an HCI task, on the other hand, are static. *Clicks* were generally uncorrelated to skill, the most interesting being the LZW complexity of a player’s clicks, with a correlation of 0.418.

Other genres and games require the player to use the mouse differently. As such, correlation of the specific features or the particular input devices is likely to be different. RTS games, for instance, are more similar to HCI tasks, in that the player uses a cursor to perform actions. It is possible that the mouse is more predictable in that genre.
CHAPTER 3. SKILL PREDICTION IN A MULTIPLAYER GAME

3.4.2 Type: event frequency, complexity and kinetics

The second grouping scheme is split into three categories: event frequency, which described how often events occurred; complexity, which measured the complexity of the player’s input using measures such as Shannon’s entropy; and kinetics, which describes the movement of the player or their input. These groups allow us to see what types of player input are most relevant to skill. There are, however, 49 ungrouped features which did not conclusively fall into any one of these categories.

As expected from the last grouping, complexity-based features that correlated well to skill described how complex the player’s keyboard input was. For example, the LZW complexity of the four movement keys (forward, left, right and back) correlates highly with skill (Pearson’s $r = 0.799$). Skilled players had a higher LZW value, implying that their input is more complex.

The kinetics group was much smaller than its counterparts. The most interesting features, corresponding to $r \approx 0.48$, include the number of times the player changed the x-direction of the mouse and the average angle of change in
That there were few well correlated features in this group suggests that this type of feature is less descriptive of player skill in an FPS. While not as prominent as keyboard, the event frequency group, contained several features that correlated highly with skill, as illustrated in Figure 3.20. In general, the higher a player’s skill, the greater number of key presses and the longer each key was pressed. In other words, players are moving for longer and reacting more rapidly, two necessary aspects in a quickly-paced multiplayer game like Red Eclipse. The advantage of these features is the relative ease with which they can be computed, another important aspect when modelling for a real-time game.

### 3.4.3 Context: Free and Dependent

If skill could be modelled independently of each game, a single model could be used to predict skill in multiple games. This would be even more transferable if this could be done without any knowledge of the internal game mechanics, but from the hardware input, which needs to be passed through the operating system. This grouping splits the features into features dependent on knowledge of the game (e.g. the user pressed a key that moves the player forward), and context-free, those features that could be extracted externally to Red Eclipse (e.g. the player pressed the ‘w’ key). The groups are relatively similar in number of features.

Although the dependent group has more features correlated with skill, several features extracted from the keyboard without know anything about the game still contained information about skill. The length of time any two keys were pressed at once, for instance, had a correlation to $\bar{s}$ of 0.780.

### 3.4.4 Player learning

In order to visualise how much players learnt over their games in Figure 3.21, we took the average score for each score group cumulatively over their subsequent games. In other words, for game number four, score was averaged over the first four games. The speed at which these values converge on their final averages, $\bar{s}$, are indicative of learning.

There is a notable increase in average performance for the two most highly skilled groups, Skilled and Expert, which is less visible in the other groups. Given
that only one person had played Red Eclipse before, this is consistent with previous research that found skill players learned faster [92].

We then selected a feature that was particularly highly correlated with player score, the average number of keys pressed at once (Pearson’s $r = 0.7896$), and plotted the cumulative average value in Figure 3.22, grouped again by score group. In contrast to Figure 3.21, the feature converges within the first three games. This has also been visualised in Figure 3.23, comparing the convergence of the two values as the correlation of their cumulative averages.

These graphs demonstrate that the feature extracted from player input is more stable than the performance measure. This may suggest that the feature is a poor indicator of skill because it is invariant to learning effects. We argue, however, that the initial learning effects present in Figure 3.21 are caused by player acclimatization to the game, rather than an increase in skill. Using Conroy’s model presented in Section 2.1.4, this could be a difference between mechanical dexterity, already present in skilled players, against prior knowledge, given that almost all players previously knew nothing about Red Eclipse.
3.5 Prediction

In this section we use the extracted features to predict several features about the player, most prominently their skill. These are separated into classification problems, predicting discrete values, and regression, predicting continuous values. In both cases we use a random forest on their default settings of \( n_{\text{tree}} = 500 \) and \( m_{\text{try}} = \lfloor \sqrt{D} \rfloor \), where \( D \) is the total number of features used for training.

![Cumulative average value for a feature over several games for each score group. For each game number, the average is taken over all games up to that point.](image1)

![The correlation of the cumulative averages to the final average for score \((s)\) and a feature of player input.](image2)
which in the experiments here are described in Table 3.7.

In these experiments, the MATLAB implementation used was an interface to the R implementation by Andy Liaw et al. [93]. Five-fold cross-validation is used, where 80% of the data is to train the classification or regression model, and the remaining used for testing. Each experiment is repeated five times and the mean accuracy or correlation reported.

The section begins by predicting skill categories, using classification, then skill measures using regression. The third section demonstrates how quickly our model can be trained through a single game. Finally, we present some preliminary analysis predicting the players’ preferences from their answers to the questionnaires.

### 3.5.1 Predicting a skill category

Some games use categories to group players by skill [13], such as StarCraft II, which uses leagues, where players in the same league are generally comparable. Classification can also be useful in single-player games, where difficulties are discrete categories (e.g. Easy, Medium and Hard). We therefore construct skill models using the score groups introduced in Table 3.5 and the number of FPSs played, \( f \). For each of these, we show separate models trained on each feature group.

The first model presented is trained to predict the score group, presented in Figure 3.24. The baseline for this model is the accuracy achieved by assigning all players to the majority class, Intermediate, with an accuracy of 27.4%. An average accuracy of 77.1% is achieved by training on keyboard features alone, significantly above the baseline. A confusion matrix is given in Table 3.9, which presents the misclassifications for a model trained on the keyboard data. Of the 131 misclassifications, 102 were in neighbouring classes, which is more acceptable for predicting skill categories than other types of classification.

For some applications, it is often sufficient to be able to distinguish between two kinds of players: those who have played before, and those who have not. For this binary classification, we split the data into two groups: Novice players in one, and all others in the second (known as 1 vs. all classification). As shown in Figure 3.25, the Context-Free group achieves an accuracy of 94.9% over a majority-class baseline of 77.2%.
3.5. PREDICTION

Figure 3.24: Mean accuracy of random forest trained to predict a player’s score group using different feature groups. Error bars indicate standard error of each model.

Table 3.9: How each game was classified for a random forest trained to predict groups of player score. Each row indicates how the examples were classified. In other words, there were 5 Novice players misclassified as Intermediate.

<table>
<thead>
<tr>
<th>prediction</th>
<th>actual</th>
<th>Novice</th>
<th>Intermediate</th>
<th>Skilled</th>
<th>Expert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Novice</td>
<td>93</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>98</td>
</tr>
<tr>
<td>Intermediate</td>
<td>15</td>
<td>64</td>
<td>19</td>
<td>20</td>
<td>118</td>
</tr>
<tr>
<td>Skilled</td>
<td>0</td>
<td>19</td>
<td>57</td>
<td>30</td>
<td>106</td>
</tr>
<tr>
<td>Expert</td>
<td>0</td>
<td>9</td>
<td>14</td>
<td>85</td>
<td>108</td>
</tr>
<tr>
<td></td>
<td>108</td>
<td>97</td>
<td>90</td>
<td>135</td>
<td></td>
</tr>
</tbody>
</table>

3.5.2 Predicting skill measures

Skill measures are continuous values that allow detailed comparison between individual players. A regression model is therefore more suitable, but has not been studied in the literature as thoroughly. Predicted measures are represented in this research with a hat. Player score, \( \bar{s} \), therefore becomes \( \hat{s} \).

As with the previous section, we constructed a model for each feature group to predict skill, \( \hat{s} \), reporting the performance for each using the magnitude of Spearman’s rank, \( |\rho| \), i.e. how similar the ranking of \( \hat{s} \) was to \( \bar{s} \). This has been summarized in Figure 3.26. The comparative baseline for this experiment is to
CHAPTER 3. SKILL PREDICTION IN A MULTIPLAYER GAME

Figure 3.25: Mean accuracy of random forest trained to detect Novice players using different feature groups. The baseline is from always guessing the player is not in the Novice category (but in one of the others instead).

Figure 3.26: Performance (Spearman’s $|\rho|$) of models trained to predict player score ($\bar{s}$). Baseline indicates mean $|\rho|$ between score ($s$) and $\bar{s}$.

use the player’s performance for the game, $s$, as a substitute. In other words, we compare how well our model predicts skill to how well using performance as a substitute would serve. $\hat{s}$ of the keyboard group correlates to the original $\bar{s}$ with $|\rho| = 0.874$, notably higher than the baseline, which has a correlation of only $|\rho| = 0.673$. This allows us to accept Hypothesis 1.

A model was also constructed to predict player accuracy, $\bar{a}$. The predicted values for this model, $\hat{a}$, have been plotted against the predicted values of player
3.5. Prediction

Figure 3.27: Relationship between predicted player score (\(\hat{s}\)) and predicted player accuracy (\(\hat{a}\)), coloured by score group.

score, \(\hat{s}\). Although the two values are not highly related (\(|\rho| = 0.675\)), there is, nonetheless, a relationship between the two models. In particular, the models tend to equally struggle to differentiate individual players, such as the more highly skilled players. It may be that the clusters created here relate to both skill and player style [27].

3.5.3 Prediction convergence rate

The features used so far have all been extracted from the entire three minutes of gameplay. However, the sooner skill can be predicted, the quicker appropriate action can be taken. We therefore extract the same features from smaller portions of the game, referred to here as segments. In addition to the full 180 s segment already used, data was extracted from the first \(t\) s of the game, where \(t \in 5, 10, 30, 60, 120, 180\).

In order to test classification speed, we split the players into two roughly equally-sized groups: Novice and Intermediate players in one group, and Skilled and Expert in the other. We then trained the model on different segment sizes, shown in Figure 3.28. This is compared to the majority-class baseline, 50.2 %, and the performance of a model trained on the player’s current score at time \(t\).

The same test has been run for a regression model, predicting \(\hat{s}\) for each segment. The performance of these models is compared to how well the current score correlates to \(\hat{s}\) in Figure 3.29. Not only are the models trained using player input
Figure 3.28: How fast a classification model is able to distinguish between two groups of players (Intermediate or worse, and Skilled or better). Dotted line indicates mean accuracy guessing the majority class.

Figure 3.29: How fast a regression model is able to predict player score ($\bar{s}$). Baseline indicates mean correlation of the current score at $t$.

more accurate than their baselines, they start to converge in a short time, (e.g. $t = 30$ s).
Table 3.10: Test accuracy (%) of random forest models trained to predict player responses based on player input to the game. Standard deviations are presented in brackets.

<table>
<thead>
<tr>
<th>Label</th>
<th>Likert</th>
<th></th>
<th>4-AFC</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Basline Accuracy (%)</td>
<td>Basline Accuracy (%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fun</td>
<td>40.0</td>
<td>37.6 (1.56)</td>
<td>49.8</td>
<td>46.4 (1.90)</td>
</tr>
<tr>
<td>Frustration</td>
<td>31.9</td>
<td>33.2 (0.822)</td>
<td>41.4</td>
<td>48.2 (1.63)</td>
</tr>
<tr>
<td>Challenge</td>
<td>34.2</td>
<td>36.9 (2.07)</td>
<td>44.2</td>
<td>45.7 (2.48)</td>
</tr>
<tr>
<td>Complexity</td>
<td>29.1</td>
<td>29.5 (2.12)</td>
<td>40.5</td>
<td>45.1 (2.36)</td>
</tr>
</tbody>
</table>

3.5.4 Predicting affective state and preference

Previous research has shown that fun, frustration and challenge can be predicted from game features [58]. We therefore briefly explore whether there is any predictable relationship between these player emotions and their input to the game.

We trained a random forest to predict each of the three reported affective states and the reported map complexity, using the full 180 s of player input data. The results of these models have been summarised in Table 3.10, which also shows the results of a model trained to predict the 4-AFC answers for each of the affective states.

Unfortunately, the results of this experiment were inconclusive. The most significant model, which predicted which game was more frustrating from a pair, was only 7 % more accurate than the majority-class baseline. We therefore do not consider there to be enough evidence to accept Hypothesis 2 in this experiment.

3.6 Summary

This chapter presented three unique analyses of player skill in the context of first-person shooters. The first is an analysis of skill measures, demonstrating the validity of particular measures over others. The results of this are summarised in Table 3.6. The most notable results are that both average player score, $\bar{s}$, and our own estimation of TrueSkill, $T$, serve as good proxies for skill in this game. In addition, when the most viable method of skill capture is through participant responses, of our two self-reported skill measures, the number of first-person shooters, $f$, that they had ever played was the most valid measure.
Using this knowledge of skill measures, we presented an extensive analysis of features extracted from player input. For an FPS, the most useful input device we found was a keyboard. Given better methods for extracting features from continuous input, however, there may be further information hidden in the player’s mouse motion. Event frequency was also notably well correlated to skill, a feature type that is somewhat easy to extract from input.

Finally, we used these features to construct a series of predictive models. These models successfully predicted player skill in a number of areas, successfully ranking players within 30 s of any game. Models were also successful at detecting players who had very little or no experience playing first-person shooters before.
Chapter 4

Automatic Difficulty Selection

Single-player games are typically presented as a campaign; an extended experience, often separated into levels, that requires the player to overcome a series of obstacles in order to, ultimately, reach the end. These are usually accompanied by a storyline that gives the obstacles some context and adds extra entertainment.

The format of a single-player campaign and its differences to a multiplayer game can create difficulties in predicting skill. The most prominent issue is that single-player games rarely have a definitive measure of performance that defines how well they did compared to others. Some score-based games exist, such as Pac-Man [94], that allow players to be compared and ranked [44]. However, the only requirement in a campaign is that the player gets to the end. In other words, there is no specification for how the players accomplish this. This makes measuring their skill more difficult.

In addition, a single-player campaign typically consists of many different types of obstacles, such as puzzles or combat situations. If skill is multi-dimensional, players will be more adept at completing certain tasks than others, making it more difficult to measure overall skill. The complexity of the skills required to complete a single-player first-person shooter has not yet been explored.

A final challenge in measuring skill is that a campaign is typically much longer than multiplayer matches. A campaign may take several hours to complete, in contrast to the few minutes that a multiplayer match may take. In order to give participants the opportunity to face different types of challenges and experience the game’s difficulty, the test bed for these experiments must reflect this.

In a single-player game, players are typically asked to select a difficulty at the beginning of a campaign. Even some multiplayer, co-operative campaigns such
as Left 4 Dead [28] give players this option. In order to select the right level of challenge, however, players need to know how hard each difficulty setting is and how good they are. This requires the player to make some form of self-assessment, which most players aren’t able to do accurately [10].

This chapter therefore presents a method for assigning a difficulty to the player using a prediction of their skill. Referred to here as automatic difficulty selection (ADS), this method also removes the need for the player to know what each difficulty setting means and works in the absence of specific measures.

DDA is an alternative, but not incompatible, strategy that improves the assigned difficulty as the player progresses. A combination of the two could have been used for this experiment, using the skill prediction to initialise the algorithm. However, this adds further complexity to the experiment and results may depend more on the chosen algorithm than the original method for difficulty selection.

In order to assign an appropriate level of challenge to the player, the difficulty of the content must first be understood. This is usually represented by some value, continuous or discrete, arbitrarily assigned by the developer to roughly represent the level of challenge for most players (e.g. Easy, Medium and Hard). However, in order to accurately assign the correct difficulty, these values must be measured in relation to the players.

This chapter is therefore separated into two experiments. The first is designed to measure skill and difficulty in a single-player game. This is then used to test whether skill can be predicted in a short time frame. The second experiment, presented in Section 4.5, explores whether or not this predicted skill can be applied in a meaningful way through automatic difficulty selection.

\section{Problem statement}

To select an appropriate level of challenge for some content and a given player, the skill of the player and the difficulty of the content must be known. While the difficulty can be worked out before release, measuring skill is more difficult in single-player games given the lack of definitive performance measure. As such, in Section 4.3, we present different methods for measuring skill in a single-player game in a first-person shooter (FPS) and analyse them in relation to each other and the content.

These methods, however, as shown in the previous chapter, require either
a long period of gameplay or the player to provide some information voluntarily. Using known skill prediction techniques from multiplayer games [13, 40], we extract features from the game to test the following hypothesis:

**Hypothesis 3** A player’s skill can be predicted in a single-player game after their first 30 s of gameplay significantly better than random guessing.

The other factor required to match challenge to the player is difficulty. It is defined in Chapter 2 as “the average performance for a piece of content over all values of skill”. This means any method for measuring difficulty should have the following properties:

1. Dependent of content
2. Independent of skill
3. Produce an expected value of challenge for a given skill

We therefore present a method for measuring difficulty and explore its adherence to each of these properties. In addition, the relationship between difficulty and the players’ skill values are compared such that an appropriate difficulty value can be selected.

Once methods for predicting skill and measuring difficulty have been found, these can be used to explore whether difficulty can be automatically selected for the player.

As mentioned, players are commonly asked to perform a self-assessment when playing single-player games by asking them to select a difficulty. However, this selection may include a degree of preference; some players would prefer to play through the campaign effortlessly, and others may relish the challenge. We therefore compare our method of prediction-based automatic selection to that of self-assessment. Self-assessment is usually the quickest way of extracting the player’s skill and removes the aforementioned preference that players may have.

In these experiments, the player’s self-assessment is referred to as $s_C$ and was collected using an adapted form of the Game Expertise Questionnaire, detailed further in Section 4.2.4. This is compared to our prediction of the player’s skill value, denoted as $s_E$, which uses the first 30 s of the player’s gameplay. The better of the two methods would assign a more suitable degree of challenge for the player. In order to test this, four methods were chosen that measured the level
of challenge present: 1) performance-based, 2) reported difficulty, 3) reported fun and 4) a representation of flow.

In a single-player campaign, performance can be measured by how far the player got through the game, or their progress. Players who experience higher challenge should not get as far as those who find the game easy. More specifically, players who experience a similar level of challenge should achieve the same amount of progress in the same amount of time. Challenge could therefore be measured objectively using the average progress of players:

**Hypothesis 4** The average difficulty and progress per second for games with a difficulty chosen using $s_E$ are at least as similar than those that use difficulty chosen using $s_C$.

To test the similarities of the performance measures, their variance are compared using a Brown–Forsythe test. This determines whether one set of measurements is significantly more spread out than another.

The second and third methods for measuring challenge used the player’s self-assessment of their experience. Players were asked to rate how hard the game was and how much fun they felt the game was. While these metrics are subjective, they more accurately represent the player’s own experience of the game than their average performance. Players may, for instance, have equally been frustrated by or enjoyed the extra challenge. These hypotheses therefore compare the success of the models using the player’s own experience:

**Hypothesis 5** The experienced difficulty of games with a difficulty chosen with $s_E$ are at not significantly different to those that use $s_C$.

**Hypothesis 6** Games with a difficulty chosen using $s_E$ are at least as fun than those with a difficulty chosen using $s_C$.

In each of these hypotheses, a Mann–Whitney $U$ test will be used to compare the average reported values of the experiment group, which uses $s_E$ (the predicted skill), to the control group, which uses $s_C$ (the self-reported skill). By accepting Hypothesis 5, we demonstrate that $s_E$ produces games that are not perceived to be any easier or harder than $s_C$. Hypothesis 6, on the other hand, can be accepted if the reported fun is as high as the control group. These will all be tested against a significance level of $\alpha = 0.01$. 
Finally, we use a representation of flow to measure how immersed players were in the game. Flow is a state of single-minded immersion which elicits feelings of enjoyment and higher performance [2]. The attention is so focussed that time appears to pass quicker for players in a state of flow. We therefore asked players to report the amount of time they thought they had been playing for, and measured flow as the difference between the reported time and actual time. Players who experienced the right level of challenge were more likely to be immersed and would therefore report a shorter time than they had actually been playing for. This is reflected in the following hypothesis:

Hypothesis 7 Players assigned a difficulty using $s_E$ experience as much flow as those using their self-assessment, $s_C$.

In order to test this, the difference between the reported and actual time taken can be calculated for each player. If the flow of the control group exceeds the flow of the experiment group according to a Mann–Whitney $U$ test, where $\alpha = 0.01$, the hypothesis must be rejected.

Testing the first hypothesis in this chapter, Hypothesis 3, is required to test the subsequent hypotheses, and thus determine the suitability of ADS. This is therefore broken down into four hypotheses, each of which can be tested separately, demonstrating the effectiveness of predictive skill in a single-player environment.

4.2 Project: Blue Room: The test-bed

The requirements for the test-bed for this experiment were similar to the requirements for the last experiment; the game needs to provide access to the source code and should be representative of other games in the genre. Although Red Eclipse provides a single-player game mode, it is relatively undeveloped and not representative of typical single-player campaigns.

The 2004 game Half-Life 2 [6] was widely well-received for its advanced facial animation, interactive storytelling and physics engine [95]. However, the Source engine, on which it was built, has been a major reason for its continued success. Alongside the game, the developers of Half-Life 2 provided users with access to the game engine and tools for content creation. These tools allow users to create their own modifications (mods) to the game that can be as extensive as fully stand-alone games. Furthermore, a community has grown around the tools that
provides additional content and helps other users. It is also possible to submit a mod online, widening the participation base. This made *Half-Life 2* a prime candidate for a test-bed.

Using these tools, we developed a short single-player game, *Project:Blue Room*.\(^1\) The game was designed to be shorter than a typical FPS so that participants were able to complete it within a couple of hours. It was, however, necessary for the game to be long enough so that players could become immersed and be observed playing a wide range of common single-player elements to accurately reflect their skill. Five levels were therefore constructed, each which had a particular overarching theme. These are described in the following section.

### 4.2.1 Levels

The game was split into five levels, each designed to represent a particular trope in single-player games. Each would also introduce new elements into the game that would be present in subsequent levels, building on the player’s experience over time as in *Half-Life 2*. Of the 15 weapons in the original game, only 5 were used in the test-bed, roughly equating to 1 weapon per level. There were also only 4 enemies used out of the total 18 available.

Each level would contain two small, blue rooms. The player would start in one and would be required to overcome a series of obstacles in order to reach the final room. Levels consisted of grey and orange textures, primarily to save time when constructing levels, but also to make the player feel like they were in a test environment.

The very first level, nicknamed *Hazard*, was designed to teach the player the basic mechanics of the game, including how to interact with the game. Some games, including *Half-Life 2*’s 1999 predecessor, *Half-Life*, include a separate tutorial that allows the player to learn the mechanics of the game separately from the campaign. More commonly these days, players are introduced to the control scheme near the start of the campaign. The aim of this research required us to monitor the player for a short amount of time. This starting level suited this purpose, and was therefore made to be relatively straightforward and possible to complete in approximately 2 minutes, but no shorter than 30 s.

The layout of *Hazard* can be seen in Figure 4.1. The rooms have been labelled,\(^1\)

---

\(^1\)The latest version of *Project:Blue Room* can be downloaded at [http://www.moddb.com/mods/projectblue-room](http://www.moddb.com/mods/projectblue-room).
4.2. PROJECT: BLUE ROOM: THE TEST-BED

Figure 4.1: How the first level, Hazard, was laid out. Key elements introduced to the player have been highlighted. Points where the player located the weapons are also shown.

Figure 4.2: At the end of Hazard, the player was asked to shoot at red targets, shown here. Destroying two targets would allow them to progress.

including the starting and ending blue rooms, as well as locations of weapons. There were no enemies in this level. Instead, players were asked to shoot breakable targets, shown in Figure 4.2.
Figure 4.3: A puzzle in the level *Introduction*. Players were required to use the given crates (brown) to get to the distant area (orange).

Figure 4.4: The puzzle shown in Figure 4.3 was changed according to the difficulty setting. Static blocks were introduced for easier difficulties to add simplicity.

The next level, *Introduction*, presented some of the basic enemies to the player and some puzzles, à la *Half-Life 2*. The puzzle elements required the player to move crates, navigate obstacles and use lateral thinking to progress. This level also contained a maze. One of the simpler examples of these puzzles is shown in Figure 4.3, which required the players to position the crates to jump to the next ledge. This puzzle was also scaled with difficulty. Figure 4.4 demonstrates how players were offered assistance for easier difficulties with static objects.
The second puzzle in this level was more challenging, as players had to work out how to get each of the crates. The number of crates was increased for easier difficulties. After encountering enemies who could shoot in a controlled environment, players were required to navigate a maze.

This led on to the third level, Combat, so-called because it was combat-oriented, requiring the player to primarily shoot at enemies in the style of traditional first-person shooters like Doom [5] and Quake [96]. This level was somewhat linear, not requiring the player to make many decisions, but get past the opponents to reach the exit. The final part of the level placed the player in a room and sent waves of enemies to attack which the player had to defeat in order to progress.

Players particularly struggled in the area labelled ‘lecture’ in Figure 4.5, because enemies had higher ground and the player was given fewer cover points. Locations of enemies have also been highlighted in this figure.

Strategy, the fourth level, allowed the player to explore the environment and use obstacles to their advantage. In contrast with the previous levels, this was non-linear, allowing the player to complete the level in a variety of ways, potentially avoiding most conflicts altogether. The size of Strategy is shown in Figure 4.7, as well as alternative routes through the level. The bridge over the middle provided a choke point for the player, but could be approached from multiple angles.

Strategy also emphasised use of alternate routes and methods for killing enemies or reaching the exit. For example, some enemies were positioned so that crates could be dropped on them, and vents were employed to give players extra options. A number of these features, including use of the crossbow, were presented to the player at the beginning of the level so they knew what to expect. There is also a rarely used feature of the Half-Life 2 engine where players can assassinate enemies with a headshot if close enough to an enemy unaware of their presence.

The fifth and final level presented the player with a ‘boss battle’, in which a player is required to defeat a single, powerful enemy, a common trope in video games. Named Arena, this level presented many of the proceeding elements in a single, climactic battle. Before fighting the boss, the player had to work their way down a few corridors and fight a handful of enemies. The arena itself was very open but provided the player with a lot of cover. The boss could only be defeated
with the rocket launcher, provided at the start of the fight. Ammo was set to
appear at randomly selected pre-determined locations, highlighted in Figure 4.8,
and indicated to the player with green lights, shown in fig:blueroom:ammo.

Upon dying, players were required to start the level from the very beginning.
Half-Life 2 normally allows players to save and load games at any point, but
this was restricted in order to simplify the experiment and provide players with
a greater degree of overall challenge.

The progress through each level was represented with a value, $0 \leq p_L \leq 100$,
that roughly described the proportion the player was through the level. This was
updated at designated points through each level. The progress through the game

Figure 4.5: The layout of the third level, Combat. Approximate locations of
enemies have been labelled. These can change according to difficulty. The room
labelled ‘waves’ has three waves of enemies, each wave increasing in number and
difficulty of opponents.
Figure 4.6: The final room in *Combat*. A set number of enemies spawn behind random doors in three waves.

was given by the total of the progress through every level, $p_G$. Given the five levels in the game, this ranged from 0 to 500, where 500 indicated the game was complete.

### 4.2.2 Difficulty on *Project:Blue Room*

The base *Half-Life 2* game uses three discrete settings which scale certain aspects of the game mechanics according to the selected setting. The most important mechanics that are scaled are the amount of damage dealt to and taken from enemies and the amount of ammo picked up. Each of these are scaled by a given factor according to the chosen setting. In addition, the amount of power the player has for sprinting and breathing underwater is also scaled slightly.

The test-bed was modified to use a single, continuous difficulty value between 0 and 1. This value was transformed using particular formulas to scaling values that scaled the above mechanics. As an example, the difficulty value, referred to as $d$, is passed through the following formula in order to determine the scaling factor for the amount of damage inflicted by the player:

$$2^{1-4d} + 0.5. \quad (4.1)$$

This means an average difficulty, where $d = 0.5$, the scaling factor is 1, i.e. damage dealt is not scaled. For the easiest difficulty, $d = 0$, the damage is
Figure 4.7: There were a variety of routes available to players for the level *Strategy*, some of which have been highlighted here. Each of the routes presented a different challenge.

multiplied by 2.5, and for the hardest difficulty, \( d = 1 \), it is scaled down by 0.625. This exponential scaling was done to accommodate the complete range of skills. The true distribution of skills was not known, so the difficulties were designed to reasonably accommodate both extremes of players. Ultimately, however, the numbers do not have any particular meaning; they are not relative to anything or anybody.

In addition to the above changes to mechanics, some puzzles were also changed according to \( d \). These were made easier or harder by removing obstacles. Unlike the previous mechanics, which were all scaled, these were changed discretely, whenever the difficulty reached a particular threshold. This was added after a
4.2. PROJECT: BLUE ROOM: THE TEST-BED

4.2.3 Experiment setup

As in the previous chapter, the experiment was conducted in-house using volunteers at the University of Manchester. In addition, however, a version of the game was released online, allowing participants to take part remotely. Both experiments collected data through a single server and some international players therefore had difficulty connecting.

For those that took part in-house, the format of the experiment was relatively

Figure 4.8: The level, Arena. The ammo points labelled were the predetermined locations for the RPG ammo. The spawn rooms are where waves of enemies would appear at regular intervals.

preliminary experiment, where participants struggled navigating puzzles.
simple. Once participants had been given a consent form they were informed that they could ask questions, as long as they were precise. Information was otherwise not volunteered. They then began playing. Participants generally played around an hour at a time, but were allowed to come back later to continue with the game if they desired, continuing from the start of the last level played. Each player was identified by a unique, anonymous identifier.

Most participants took part in the same environment, using the same machine, mouse and keyboard. Eight of the participants, however, played simultaneously on a different group of machines. These machines were higher powered than the test machine, but participants were still required to observe the same format, and were observed at all times.

For online participants, there was no given structure; they were free to download and play the game at any time. The game was instrumented, however, to make cheating more difficult. Where players were detected cheating, their games were removed from the experiment. Online participants were identified by their Steam ID, a unique, semi-anonymous identifier, that can only be tied to information the player has chosen to make publicly available.

Using the information logged by the game, a summary of each game’s statistics were made available, accessible using the player’s identifier. Examples of statistics presented included the difficulty, the time taken and the number of enemies killed for each level. Presenting these personalised statistics to players was designed to encourage competitiveness and reward players for participating. However,
participants were only presented with the appropriate link upon completing the game.

As with *Red Eclipse*, the game engine was instrumented to log events. However, this game, unlike the previous experiment, which consisted of small, compartmentalised sessions that were closely monitored, was much more open-ended and needed to account for players who participated online. The game engine was therefore modified to communicate directly with the remote server, not only while logging events, but also when changing levels, recording a hash of each level played to ensure players played the correct sequence of levels. The game’s console was also disabled to prevent players from changing game settings or cheating. Finally, loading was disabled, so players were not able to restore saved games. The only given control the players had over the game was to either start a new game or restart the current level at any point.

While the majority of events recorded in *Red Eclipse* were based on the player’s input, the events in this experiment more extensively covered higher-level game events. The complete list of events has been described in Table 4.1. Most notable are the absence of mouse events, which were less informative in the last experiment and had large storage requirements. Omitting these events greatly reduced bandwidth requirements.

Each event also included a time stamp and 3-dimensional co-ordinate representing the player’s position in the world, allowing us to reconstruct the player’s movement through their game.

Other choices were partially limited by the engine used. KeyUp events, for instance, were only fired for a selection of associated key down events, such as movement. In this experiment, kills and damage between non-playable entities (enemies) are recorded. However, the game does not record some enemy events, such as their attacks or their reloads.

For this experiment a player was assigned a uniformly random difficulty whenever they started a new game. Players could therefore start a new game in order to experience a different difficulty. The two objectives for this experiment were to measure player skill, in order to construct a predictive model, but also to objectively measure game difficulty.

Players were initially not asked to complete any questions. This avoided taking the player out of immersion, and meant there was no requirement for online participants to have to provide feedback. This was done because the
### Table 4.1: The events recorded in *Project: Blue Room.*

<table>
<thead>
<tr>
<th>Name</th>
<th>Fired when...</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyDown</td>
<td>The player presses a key, code, that is bound to some action, <code>binding</code>.</td>
<td><code>binding</code> is a string registered to a particular action, e.g. <code>+moveleft</code>.</td>
</tr>
<tr>
<td>KeyUp</td>
<td>A key is released.</td>
<td>This is only fired if the associated KeyDown event expects a key release, e.g. <code>+moveleft</code>.</td>
</tr>
<tr>
<td>Death</td>
<td>The player or an enemy dies.</td>
<td>Records who killed whom.</td>
</tr>
<tr>
<td>Damage</td>
<td>A player or enemy receives damage.</td>
<td>In addition to who was involved, this also records the type of damage that was dealt and how much.</td>
</tr>
<tr>
<td>HealthPickup</td>
<td>The player picks up a health kit.</td>
<td>Records the amount of health given and the health remaining, but also how much was actually useful (given the player cannot have more than 100 health).</td>
</tr>
<tr>
<td>AmmoPickup</td>
<td>The player picks up ammunition.</td>
<td>As with HealthPickup, records the amount and the ‘effective’ amount.</td>
</tr>
<tr>
<td>WeaponPickup</td>
<td>A weapon is picked up.</td>
<td>In <em>Project: Blue Room</em> it is only fired when the player picks up a new weapon.</td>
</tr>
<tr>
<td>WeaponSelect</td>
<td>The player selects a different weapon.</td>
<td></td>
</tr>
<tr>
<td>PlayerShoot</td>
<td>The player fires.</td>
<td>Describes the weapon and ammunition used, along with how much ammunition is left.</td>
</tr>
<tr>
<td>PlayerReload</td>
<td>The player reloads.</td>
<td>In addition to the weapon and ammunition, this event records how much ammunition was left in the clip before reloading.</td>
</tr>
<tr>
<td>Progress</td>
<td>The player reaches a progress point.</td>
<td>The progress points are associated with a numerical value that indicates the player’s progress through the level as a percentage.</td>
</tr>
</tbody>
</table>

Best skill measures in the last experiment were calculated from in-game data. However, following the experiment, there were insufficient data. As such, an
adapted form of the Game Expertise Questionnaire was asked, described below. In-house participants who completed this questionnaire were offered a £5 Amazon voucher, while online participants were entered into a prize draw for a £20 voucher upon completion.

### 4.2.4 The adapted Game Expertise Questionnaire

An adapted form of the Game Expertise Questionnaire (GEQ) was used in these experiments that built on the results of the multiplayer-based experiment. The adaptations were made in order to reduce the overall length by removing less relevant questions and asking more questions that better related to our research.

The original GEQ separated questions into genres, asking participants to answer each with respect to the genre. All genres except the FPS genre were removed, refining the focus of the questionnaire.

Given its success in Chapter 3, a question was added that asked the players how many first-person shooters they had played before. Selecting ‘None’ disabled the remaining answers, thereby reducing the time of the questionnaire for some players.

In the previous experiment, some participants reported being unclear as to whether to count their experience on console games or only report the experience relevant to PC gaming. In order to capture this, participants were asked how their time was distributed between PC games and console games, potentially allowing for further analysis on the differences in players. Participants were also asked to rate the amount of experience in single-player against multiplayer games, similarly capturing the differences in skills that might arise. There is no known research which explores the difference in skills between different input devices or the differences in single or multiplayer games. These questions were therefore designed to account for any differences that may exist while keeping the questionnaire relatively short.

The GEQ also asks players to report whether they had ever played an FPS for more than 5 hours a week. This question gave less opportunity for analysis than others, and was therefore removed. Instead, participants were asked to report how many hours, in total, they had ever played. This followed from previous research, which has shown that the accumulated practice of an individual is directly related to their skill [97].

The final thing that the GEQ asks of participants is to circle extensively played
games for a given genre, or list others. In order to reduce priming effects caused by suggesting games, we instead asked the players to list them from memory.

In summary, participants were presented with the following questions:

1. How many **years** have you been playing video games?

2. How **many** first-person shooters had you played before *Project: Blue Room*?
   - None, 1 or 2, 3 - 6, 7 - 12, or 13+

3. How would you rate your **expertise** level for your **most played** first-person shooter?
   - (Never played) 0 – 5 (Expert)

4. How many **hours** have you played first-person shooters **in total**?

5. How many **hours per week** have you played first-person shooters in the **last six months**?
   - None, 1 or 2, 3 - 6, 7 - 12, or 13+

6. How much time have you spent playing first-person shooters on a **PC** vs. on a **console**?
   - (Exclusively PC) 0 – 5 (Exclusively console)

7. From your experience of first-person shooters, how much has been of **single-player** games vs. **multiplayer** games?
   - (Only single-player games) 0 – 5 (Only multiplayer games)

8. Please list any first-person shooters you have played **extensively**.

The questions with a Likert [87] scale (0 – 5 above) were originally 7-point questions. These were changed to 6-point in order to prevent players defaulting to a middle value and force a preference.

### 4.2.5 Data distribution

187 games from 187 players were used in this experiment, 36 of whom participated in-house. These games totalled 128 hours of overall playing time, separated over 1,236 lives. The mean game progress was $\bar{p}_G = 366$, or 73.2 %. 69 players completed the game, and had an average completion time of 44 minutes.
Figure 4.10: Average game progress, $p_G$, and time playing $t_G$, for participants who took part in-house against those who took part online.

This sample of players, used in the following research, was taken from a larger data set. The complete data set (described here for completeness), consisted of 260 participants and 317 games, which accounted for 166 hours worth of data. These games were then filtered down so that only games of consequence were included. 5 players were removed because they cheated or pressed unexpected keys. A few games were also removed because they did nothing for several minutes. Only the first game for each player was used in order to minimize bias towards players who played several times. Of the remaining games, those that were complete, had more than one life or were longer than 30 minutes were kept. This heuristic was designed in order to remove games for which the player was deemed to have not tried hard enough.

Most in-house participants took part in a single session, which typically lasted around an hour. In contrast, online participants, on average, played for closer to 30 minutes. This difference in playing time is contrasted in Figure 4.10. There was, however, no statistical difference in progress between the two groups.

91 participants completed the adapted game expertise questionnaire. The players’ answers to the FPSs played question has been shown in Figure 4.11. For each of these figures, in-house participants are highlighted. While these players were relatively evenly distributed, the majority of online participants

---

2Some players used commands that altered the state of the game, for instance slowing down time. These commands were bound to key presses and could therefore be detected.
had previously played a significant number of first-person shooters. Given online participants had to own *Half-Life 2* in order to participate, this was as expected. This was also reflected by their reported expertise, shown in Figure 4.12. Online participants generally reported greater skill. No participants reported having no previous experience at FPS games for this experiment. To contrast these, the reported number of hours played is shown in Figure 4.13. In-house participants generally reported playing very few hours per week compared to their online counterparts.

### 4.3 Measuring difficulty and performance

Unlike multiplayer games, where a particular task can be used to measure performance, single-player games rarely provide an absolute ranking of players. If they did, then players would likely change their style of playing. Research has previously demonstrated players generally exhibit different playing styles [47]. These different styles make it harder to measure absolute performance in a single-player game, and subsequently skill and difficulty.

This section therefore uses the performance measures and questionnaire responses to evaluate different methods for measuring skill and difficulty in the context of a single-player game.
4.3. MEASURING DIFFICULTY AND PERFORMANCE

Figure 4.12: The distribution of reported expertise, $e$, for this experiment.

Figure 4.13: The distribution of average hours played per week in the last six months, $h$, for this experiment.

### 4.3.1 Performance

Some single-player games offer a particular objective to the player, e.g. completing a quest, or vanquishing a particular enemy. How the player chooses to accomplish this, however, is up to them. Some games will even provide the player with multiple objectives and allow the player to pursue any of them, or none at all. The *Elder Scroll* series, for example, provides the player with a ‘main quest’ and multiple side quests [98]. Some players will never bother completing the main quest.
In an FPS campaign, the implied objective is to complete the game. This is therefore a binary measure of performance. The two main issues with this measure are its reliance on numerous factors, including the player’s motivation and the game’s difficulty, and that it is not very descriptive. Even adjusting for difficulty, the most that this measure can rank players is stating that players with a difficulty, \(d \geq a\), that complete the game are better than those with difficulty of \(d \leq a\) that do not.

A more generic measure of performance than completion success is the progress, where, for Project:Blue Room, \(p_G = 500\) is equivalent to completion. This allows us to compare the success of players with more precision. Players that have a higher \(p_G\) are likely to be more skilled than others. Unfortunately, this is still reliant on factors such as the game’s difficulty and the player’s motivation. A low \(p_G\) value, for instance, may indicate boredom.

Other common measures of performance include the number of times the player dies (or, alternatively, their lives), the number of enemies killed, or the time taken to complete the game, \(t_G\), [46]. Some games that use DDA also use the player’s current health [29], as it is an indicator for how likely they are to die. However, each of these performance measures may be subject to a player’s preferred playing style.

We use two performance measures in our experiment, game progress, \(p_G\), and game time, \(t_G\). Unfortunately, players who do not complete the game will report a lower time than those who have. We therefore use the completed time, \(t_C\), or the time taken for players who completed the game. While this does not allow us to rank players who did not complete the game, it gives a ranking for players where \(p_G = 500\) and therefore cannot be ranked with \(p_G\).

These measures are, however, influenced by the difficulty of the game, as shown in Figure 4.14. \(p_G\) has a Pearson’s \(r = -0.524\), and \(t_C\) of \(r = 0.295\). Unlike a multiplayer game, where performance can be averaged over several games, players typically only play a game once. Those that play a second time will find it easier. Therefore, each player was assumed to have played long enough for the performance measure to be representative of their skill, and this performance measure was then converted to a percentile relative to other players who played a similar difficulty.

For a group of \(N\) players who played on the same difficulty, the skill of the player, \(s_i\), can be given as the player’s ranking relative to everyone else in the
Figure 4.14: The relationship between two performance measures, game progress, $p_G$ and completion time, $t_C$, to difficulty, $d$. Standard error is indicated for each measure ($\frac{\sigma}{\sqrt{N}}$).

group for a given performance measure, $p$:

$$s_i = \frac{\sum_j [p_j \leq p_i]}{N}.$$ (4.2)

The notation $[A]$ is known as Iverson notation [99] and is given by:

$$[A] = \begin{cases} 
0 & \text{if } A \text{ is true;} \\
1 & \text{otherwise.}
\end{cases} \quad (4.3)$$

This method can be applied to any content and any measure of performance. However, it is only a heuristic of skill. This can be generalised to account for varying difficulty by measuring the ‘distance’ of the difficulties between two players. The more different another difficulty it is, the less relevant. For this experiment, a Gaussian function was used for weighting:

$$s_i = \frac{\sum_j [p_j \leq p_i] f(d_i - d_j)}{N},$$ (4.4)
where

\[ f(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}, \]  

(4.5)

\( d_i \) is the difficulty for player \( i \), and \( \sigma \) changes how evenly the weighting is spread. Larger values of \( \sigma \) put more emphasis on further difficulties.

The adjusted forms of the measures \( p_G \) and \( t_C \) are used as \( \tilde{p}_G \) and \( \tilde{t}_C \) respectively. The summary of skill measures used in this experiment is summarised in Table 4.2. The moving average of these measures with a window size of 0.2 is shown in Figure 4.15. Although the correlation to difficulty is still present, it is lower at Pearson’s \( r = -0.231 \) for \( \tilde{p}_G \) and \( r = 0.0446 \) for \( \tilde{t}_C \).

The next two measures of skill used were taken from the adapted game expertise questionnaire: reported expertise, \( e \), and FPSs played, \( f \). The game progress, \( p_G \) for each for these is shown in Figure 4.16 and Figure 4.17 respectively. Unfortunately, the number of players who had completed the game and responded to the questionnaire were too sparse for lower levels of skill, so \( t_C \) is not shown. Both of these graphs display large ranges of performance for some skill values. Two major factors of this are the random difficulties assigned and the players’ poor self-assessments.

In summary, four methods for measuring skill in a single-player game are presented; two based on performance measures and two self-reported. The two
4.3. MEASURING DIFFICULTY AND PERFORMANCE

Table 4.2: A summary of performance measures in *Project: Blue Room*.

<table>
<thead>
<tr>
<th>Name</th>
<th>Games</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjusted progress</td>
<td>$\bar{p}_G$</td>
<td>187 The player’s progress through the game relative to games of a similar difficulty.</td>
</tr>
<tr>
<td>Adjusted time</td>
<td>$\bar{t}_C$</td>
<td>69 The total time taken for completed games, relative to games of a similar difficulty.</td>
</tr>
<tr>
<td>Reported expertise</td>
<td>$e$</td>
<td>91 The player’s self-reported assessment of skill.</td>
</tr>
<tr>
<td>FPSs played</td>
<td>$f$</td>
<td>91 The number of FPSs the player reported they had ever played.</td>
</tr>
</tbody>
</table>

Figure 4.16: Average game progress, $p_G$, for each expertise group. There were no players in this experiment that reported having an expertise of $e = 0$.

Performance-based measures, $\bar{p}_G$ and $\bar{t}_C$, were somewhat dependent on difficulty and too noisy to use in a real world environment. In particular, the adjusted completion time, $\bar{t}_C$, requires players to complete the game, omitting a large portion of data.

Of the two self-reported measures, reported expertise, $e$, was chosen due to its slightly larger range and more even distribution of players, as seen in Figure 4.12.
4.3.2 Difficulty

Difficulty is usually described by some arbitrary number, which may be from a set of values, e.g. $d \in \{1, 2, 3\}$, indicating three difficulty settings. However, these values are unlikely to hold any further meaning. Does a difficulty value of $d = 2$, for instance, indicate the game is twice as hard as $d = 1$? If this is the case, what does it mean for something to be ‘twice as hard’? Is it that half as many people can succeed, or that twice the performance is required?

The method presented here for measuring difficulty uses the distribution of player performances of the continuous difficulty setting, $d$, where $d = \{x \in \mathbb{R} | 0 \leq x \leq 1\}$. It is also demonstrated that this method follows the following three requirements:

1. It must reflect the differences in content
2. It must be independent of skill
3. It must be able to produce an expected value of challenge for a given skill and a given difficulty.

For a piece of content (in this case a level) and a range of difficulty, the total number of attempts and the number of completed attempts are known. This means that for any particular range of difficulty the probability of completing the
level can be calculated. Figure 4.18 shows these values for players of the second level, Introduction.

Harder difficulties have more overall attempts and fewer relative completed attempts. In other words, the proportion of completed games is lower. This proportion is equivalent to the probability of completing the attempt, visualised in Figure 4.19 for the second level, Introduction. For the most part, the chance of completion in this level is relatively uniform, decreasing for values of $d$ greater than 0.6. In contrast, the following level, Combat, shows a greater probability of completion for $d < 0.3$ in Figure 4.20. However, the difficulty quickly increases, dropping to lower than 0.1 in the range $0.9 \leq d \leq 1.0$. The difference between these levels demonstrates that this method of measuring difficulty is dependent on content, satisfying the first requirement.

The second requirement, that the difficulty should be independent of skill, is satisfied by using the population of players. A better method for calculating difficulty from skill can be obtained by adjusting the expected probability of completion to equal the player’s relative skill. So novice players should have a 100% chance of completing this, and the best player in the world would have close to 0% chance.

Let the relative skill ($s_r$) of the player be represented by the proportion of players with a skill measurement greater than that of this player, i.e. $s_r = \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$. Find the difficulty value, $d$, such that the proportion of
CHAPTER 4. AUTOMATIC DIFFICULTY SELECTION

Figure 4.19: The proportion of completed attempts over difficulty, $d$, for the level Introduction. As difficulty increases, attempts are less likely to be completed.

Figure 4.20: The proportion of completed attempts over difficulty, $d$, for the level Combat. Difficulty drastically increases for values of $d > 0.4$.

the difficulty distribution is less than or equal to $s$. As an example, Figure 4.21 demonstrates how appropriate values for $d$ can be selected for skill values, $s = 0.5$ and $s = 0.8$. It is clear that the harder level, Arena, is assigned significantly lower values of $d$. It can be shown that this procedure causes the expected probability of completion to equal relative skill$^3$.

$^3$According to [100] an expectation can be calculated from a cumulative distribution function ($F(x)$) based on the following equation: $E(x) = \int_{0}^{\infty} 1 - F(x) \, dx$. Using this insight, we can express the expected completion probability as $E(C) = \int_{0}^{d_r} 1 - P(C \mid d) \, dd$, where $d_r$ is the
4.4. PREDICTING SKILL

This section explores using features extracted in the single-player game, Project:Blue Room, to predict various measures of skill. We first present the features, extracted from the player input and higher-level game events, and then demonstrate how these were used to predict skill.

---

Figure 4.21: Cumulative probability of completion for two levels, Introduction and Arena, demonstrating selecting difficulty values, \( d \), for different skill measures, \( s \). Mapping \( s = 0.5 \) and \( s = 0.8 \) produces \( d = 0.458 \) and \( d = 0.745 \) respectively for Introduction, and \( d = 0.205 \) and \( d = 0.478 \) for Arena. In each case, the cumulative probability has been normalised to between 0 and 1.

A major advantage of this method is that it is generalisable, both to different skill measures, transforming them to appropriate an appropriate range, and different content or games. This method is also objective, not requiring developers to record feedback from players or manually analyse game logs. However, the main drawback is that it requires a large sample in order to appropriately represent the population. In this experiment, the number of game used was assumed to be great enough. This method also struggles where the level of challenge is too low, as it only scales difficulty values, \( d \), down. In the worst case, where the proportion of completed games is uniform, \( d = s \).

---

required difficulty. This should equal the proportion of players better than the current player, which is \( E(c) = 1 - s_r \). Rearranging yields \( s_r = \int_0^d \text{P}(C|d) \text{d}d \) which is the procedure described in the main text.
4.4.1 Features

Unlike the last experiment, which used player input alone to predict skill, the features extracted included game features, taking advantage of the greater range of higher-level interactions recorded in Project:Blue Room.

Of the features extracted from Red Eclipse, 24 of the most interesting and least complex features were used. Unlike the last experiment, however, which focused on player input alone, an additional 24 features were extracted from the higher-level events recorded in Project:Blue Room.

These features generally described one of four aspects: the amount of damage dealt, the amount of damage received, the ammunition (ammo) used and the player’s shooting habits. Two other features, the average health on each event and the player’s accuracy, were also extracted because they are commonly used as heuristics for the player’s performance. Each feature was designed to capture how efficient the player was or how much challenge they were facing, in some cases both. Some features were also extracted that described how fast the player responded to danger, such as the average time between taking damage and damaging the opponent, heuristics used in previous research [27].

In addition to these, six generic features were also extracted from each type of event:

- The number of events per second
- The standard deviation of events sampled per second
- The number of event ‘bursts’ per second
- The LZW complexity of the events
- The maximum time between any two events
- The standard deviation of times between events

Some of these features were removed due to their similarities. In other words, using them separately did not provide any further information about the players. The number of events per second for KeyUp events, for instance, very highly correlated with the number of events per second for KeyDown events (Pearson’s $r = 0.985$). These redundant features were removed before proceeding. There were 63 features of this type remaining, giving a total of 111 features to use in the
4.4. PREDICTING SKILL

64 of the more interesting features have been listed and described in Appendix C.

4.4.2 Prediction

In order to demonstrate skill prediction in a single-player game, we use three models. The first is an LDA model, which is solved using singular value decomposition (SVD). A random forest was used with the same settings as in Section 3.5, i.e. $\text{ntree} = 500$ and $\text{mtry} = \lfloor \sqrt{D} \rfloor$. An SVM was also used, or, more specifically, a support vector regression model (SVR). The parameters used for this model were $C = 1.5$, $\epsilon = 0.2$ and $\gamma = 0.25$, although we found the model was largely insensitive to parameters. Although more difficult to tune and more sensitive to features, this model was faster than a random forest and more compact, only needing to store the support vectors. All three models used in this section were from the scikit-learn library for Python [101].

Each experiment separates the data into five portions for five-fold cross-validation, 80% of the data used for training, and 20% for testing. For these experiments, only games that provided a response to the questionnaire are used. In addition, prediction of the skill measures are only done from the first 30 s of each player’s first level.

Initially, a random forest model was trained to predict player expertise from the first 30 s of gameplay. This achieved a testing accuracy of only 37.3%. Given the majority class baseline is 39.6%, this appears no better than random guessing. A confusion matrix of the model, however, shown in Table 4.3, demonstrates that the guesses are not evenly distributed. In this case, 65.8% of misclassifications are in neighbouring classes. A regression model also finds a certain degree of correlation at $\rho = 0.218$.

In order to improve this accuracy, an LDA model was employed to further reduce these features into something more manageable. Using the first 30 s of every game by players who answered the questionnaire, a model was trained to separate the features using the reported expertise, $e$. The first two dimensions produced from this are displayed in Figure 4.22 for the first level, Hazard. There is a great deal of overlap for the average group, $e = 3$. However, there are clearer distinctions for the separations between extreme groups. The first dimension, LDA 1, is correlated to $e$ with Spearman’s $\rho = 0.474$. This uncertainty of the model, particularly for the more average values, such as $e = 3$ and $e = 4$, could
Table 4.3: The classifications for a random forest trained to predict reported expertise \((e)\) for a portion of the data set, presented in a confusion matrix. The numbers presented here are answers to the Likert questionnaire regarding expertise. As before, rows indicate actual value (two players with \(e = 1\) were misclassified).

<table>
<thead>
<tr>
<th>prediction</th>
<th>actual</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>6</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>23</td>
<td>16</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.22: The first two dimensions of an LDA model trained on reported expertise, \(e\).

be down to the Dunning-Kruger effect, where participants are misreporting their skill, or to the more numerous aspects of skill required in a single-player game.

Using this model, a wrapper method was employed to reduce the features used and determine the utility of each feature. For this, an SVR model was used to learn the reported expertise, \(e\), chosen due to its speed over random forests, which is important in real-time games.

The least informative features were sequentially removed in a process known as
4.4. PREDICTING SKILL

Figure 4.23: How performance of an SVR model trained to predict reported expertise, $e$, improves with the removal of given features.

recursive feature elimination. For each step, both models were trained on the same portion of the data and tested on the rest in 5-fold cross-validation. This process was also repeated 200 times over random shuffles of 5-fold cross-validation for each feature, removing the feature that contributed the least on average. The results for this are shown in Figure 4.23. The best performance, Spearman’s $\rho = 0.431$ was achieved using the last 16 features, described in Table 4.4. The complete list of features used for this is given in Appendix C.

Some of these features, such as the number of key presses, are expected given the last experiment. However, some higher-level game events are present, most notably the Progress and WeaponPickup events, features that were extracted from the first 30 s of the first level. As such, players who reached the first progress point in that time would be more skilled. The first and third features listed in Table 4.4 are visualised in Figure 4.24. The feature based on the game event, $\text{progress-maxtime}$, is less descriptive than its counterpart, $\text{numdiffkeys}$, and depends on the content; while $\text{numdiffkeys}$ can be used in other games, $\text{progress-maxtime}$ requires the game to use Progress events and to include one near the start of the level.

Interestingly, the difference in the Progress-based feature is more pronounced for players of higher skill, while the player input-based feature better differentiates between players for $e \leq 3$. This may imply that expertise at the controls, or ‘mechanical dexterity’, is more quickly mastered than higher-level abilities, such
Table 4.4: The final 16 features used by the SVR model to predict reported expertise, $e$, ordered by decreasing importance. The correlation given represents the model’s accuracy after removing the given feature in the reverse order given. The complete list is presented in Appendix C.

| Name              | Description                                                                 | Spearman’s $|\rho|$ |
|-------------------|-----------------------------------------------------------------------------|------------|
| numdiffkeys       | The total number of different keys the player uses.                        | -          |
| persec-movement   | The total distance moved by the player per second.                          | 0.271      |
| progress-maxtime  | The maximum time between any two Progress events.                           | 0.301      |
| weaponpickup-bursts | The average number of WeaponPickup bursts per second.                     | 0.284      |
| mean-persec-keyspressed | The average number of key events per second time window, after grouping events to the nearest second. | 0.308      |
| num-crowbar       | The number of times the crowbar was swung.                                 | 0.334      |
| keyup-lz-bin      | The LZW complexity of KeyUp events.                                        | 0.346      |
| progress-std      | The standard deviation of number of Progress events per second.            | 0.367      |
| weaponpickup-lz-num | The LZW complexity of WeaponPickup events.                               | 0.373      |
| shoot-lz-num      | The LZW complexity of Shoot events.                                        | 0.395      |
| all-std           | The standard deviation of number of events per second, grouping events into second clusters. | 0.411      |
| persec-movementkeys | The total time the player held the movement keys per second.             | 0.417      |
| keytime-requiredkeys | The total time the player held any of the ‘required’ keys (+forward, +moveleft, +moveright, +back, +attack, +use, +jump and +duck). | 0.395      |
| shoot-std         | The standard deviation of Shoot events per second.                         | 0.404      |
| lz-forwards       | The LZW complexity of the player’s forward key presses (represented as a binary string). | 0.421      |
| freqentropy-command | The Shannon entropy of the player’s key presses.                     | 0.428      |
4.5 Applying skill capture to difficulty selection

Once a skill model had been constructed for Project:Blue Room and a method of selecting an appropriate difficulty had been chosen, the next step was to explore the success of its application. For this, we performed a second experiment using Project:Blue Room, described in this section.

4.5.1 Experimental setup

In order to test ADS, two test groups were required: the experiment group, $E$, who were assigned a predicted skill measure from the first 30 s of their gameplay, and the control group, $C$, whose skill was assigned from their own self-assessment. The mapping of these skill values to difficulty were the same for each group, and has been described below.

For the control group, who reported an expertise value, $e \in 0, 1, 2, 3, 4, 5$, the skill value was transformed to a skill value $0 \leq s_C \leq 1$. Rather than mapping the lowest $e$ value to the lowest $s$ value, we assumed each group was normally distributed. Therefore, skills were assigned in bands, where the $s_e$ value fell in the middle of that band.

Figure 4.24: Two features grouped by reported expertise, $e$; numdiffkeys, which describes player input, and progress-maxtime, which describes the speed at which they complete the first challenge.

as problem solving. This is discussed further in Section 5.2.3.
Players in the experiment group were assigned a predicted value of their expertise, $\hat{e}$, using the predictive model above. Unfortunately, because very few players rated themselves as having low skill, we needed to adjust for the model’s bias. We therefore adjusted $\hat{e}$ to between $p_{\text{min}}$ and $p_{\text{max}}$, limits chosen using the $\hat{e}$ values assigned to the lowest and highest skill groups $e = 1$ and $e = 5$ (no-one in the first experiment reported having an expertise of $e = 0$). In order to do that, we assumed that the predicted values for each expertise group was normally distributed, and selected $p_{\text{min}}$ as one standard deviation below the mean for group $e = 1$ and $p_{\text{max}}$ as one standard deviation above the mean for group $e = 5$. This method coped with any outliers, and encompassed the majority of players within each group, as visualised in Figure 4.25.

\begin{align}
s_C &= \frac{e + 0.5}{6}, \quad (4.6) \\
\hat{s}_E &= \frac{\hat{e} - p_{\text{min}}}{p_{\text{max}}}, \quad (4.7)
\end{align}

where

\begin{align}
p_{\text{min}} &= \hat{\mu}_1 - \sigma_1, \quad (4.8) \\
p_{\text{max}} &= \hat{\mu}_5 + \sigma_5. \quad (4.9)
\end{align}
These skill values, $s_E$ and $s_C$, were interpreted as percentiles of the population, where 0 indicated there were no players of a worse or equal skill, and 1 indicated there were no players better. These values were then transformed into a difficulty for each level.

As described in Section 4.3.2, each level was represented by the probability of completion for each difficulty value, $d$. The skill, $s$, was then mapped to the value $d$ such that the percentile of the distribution was less than or equal to $s$, as visualised in Figure 4.21.

In order to collect the data for measuring challenge, players were asked to complete a series of questionnaires. The first questionnaire was the adapted Game Expertise Questionnaire described in Section 4.2.4, from which $s_C$ was calculated. The player was then asked to answer two questions after each level. These were presented on a paper sheet that listed all levels with their names and the following two questions for each level:

- How **difficult** was that level? Likert scale: (Too easy) 0 - 5 (Too hard)
- How **fun** was that level? Likert scale: (Not fun) 0 - 5 (Very fun)

Upon completing the game or reaching the end of the two hour experiment, participants completed a final online questionnaire that quizzed them on their overall experience of the game:

- How **difficult** was the game overall? Likert scale: (Too easy) 0 - 5 (Too hard)
- How **fun** did you find the game overall? Likert scale: (Not fun) 0 - 5 (Very fun)
- How many **minutes** do you think you have been playing for?
- Please **order** the game’s levels by **difficulty**.
- Please **order** the game’s levels by **fun**.

The final two questions, which asked players to rank the levels, were equivalent to a forced alternative choice. For these questions, players were presented with images and names for each level and asked to drag them into an appropriate order.
The format of the experiment has been presented in Figure 4.26. Other than the structure of the experiment and the assigned difficulties, the experiments also lasted, at most, two hours. Four minutes before the two hour mark, players were asked to quit the game, answer the level questions for the level they were on and fill in the final questionnaire.

In order to encourage participation, 40 players were offered a £10 Amazon voucher upon completion of the experiment. In all other respects, this experiment was identical to the in-house experiment presented in Section 4.2.3.

### 4.5.2 Data distribution

As with the previous experiments, we present how the data was distributed in terms of the Game Expertise Questionnaire. There were 40 participants in total, 21 of which were in the experiment group, 19 in the control group.

Figure 4.27 shows that there is a somewhat even distribution of experience in terms of years playing video games. Half of the participants had played games for between 13 and 20 years. Half of the remaining participants had only played for 7 years or less.

The number of first-person shooters played, $f$, is visualised in Figure 4.28. This indicates that most of our participants were relatively familiar with first-person shooters. This is backed up by Figure 4.29, in which most players reported having a skill of 3 or greater.

As with previous experiments, we found although reportedly relatively skilled, most participants had not played many video games within the last six months (Figure 4.30).

Finally, comparing the average expertise in the control group to the expertise group with a Mann–Whitney $U$ test indicated no statistical difference ($p = 0.483$).
4.5. APPLYING SKILL CAPTURE TO DIFFICULTY SELECTION

Figure 4.27: The distribution of the number of years played by each participant in Phase II of Project:Blue Room.

Figure 4.28: The distribution of the number of first-person shooters played by each participant in Phase II of Project:Blue Room.

4.5.3 Results

Once the experiment was complete, the data from the two groups were compared in order to test the hypotheses outlined at the start of this chapter.

The purpose of this experiment was to compare the level of challenge for each of the two groups, C and E. The difficulty assigned to each of the groups for the second level, Introduction, is shown in Figure 4.31, separated by their reported expertise, e. Participants in group C received a static difficulty directly related
to their reported expertise. The experiment group, however, were assigned their difficulties based on their predicted skill.

From Figure 4.31, it is visually clear that participants in group $E$ received a generally increasing difficulty with their reported expertise, $e$. Indeed, there is a Spearman’s $\rho$ correlation of 0.753 between $e$ and the assigned difficulty.

For most hypotheses, the Mann–Whitney $U$ test was sufficient, comparing whether or not the values of the two were different. For Hypothesis 4, however,
we required a way of measuring the spread of the performance values. Figure 4.33 shows the average progress per second for each group and the actual difficulties assigned. A Mann–Whitney $U$ test of the two groups states that we cannot assume that the two groups are drawn from different samples. The first group has a standard deviation of $\sigma = 0.0557$, the second $\sigma = 0.0379$. A Brown–Forsythe test concludes that there is not enough evidence to assume the two groups have different spreads. In other words, the level of challenge provided to players in group $C$ are no more or less varied than those in group $E$.

The average answers for difficulty and fun for each of the test groups is visualised in Figure 4.34. There is no significant difference in reported difficulty and fun according to a Mann–Whitney $U$ test ($p = 0.0856$ and $p = 0.272$ respectively). Interestingly, however, there is a significant difference in actual difficulty ($p = 0.00686$). In other words, players performed the same and reported the same level of difficulty, but one group was assigned a significantly harder difficulty. These results do not provide enough evidence to reject the null hypothesis for Hypothesis 5 or Hypothesis 6 (that predicted values of skill are as good as self-reported measures). This research therefore indicates they may be accepted.

Finally, players were asked to report the amount of time they thought they had been playing for. These values are compared to the actual playing time for each test group in Figure 4.35. The centre line indicates where the player’s prediction would match the actual time. In order to test Hypothesis 7, the difference of
Figure 4.32: The average progress made per second for each of the expertise groups, $e$ in each of the test groups.

Figure 4.33: The average progress made per second for each of the test groups.

each point from that line is calculated, visualised in Figure 4.36. In other words, the line $x = y$ in Figure 4.35 has been transposed to $y = 0$ in Figure 4.36, demonstrating how well the playing time was estimated. The Mann–Whitney $U$ value for these two groups is $U = 125$. For $\alpha = 0.01$ ($p = 0.0350$), there is again insufficient evidence to confidently state that the amount of flow experienced by the control group was greater than that in the experiment group. This research therefore accepts Hypothesis 7.
4.6 Summary

This chapter has presented a thorough approach to automatically selecting a player’s difficulty in a single-player campaign after only 30 s of gameplay. Unlike the previous research in multiplayer games, these results were somewhat less conclusive, largely down to the problems outlined at the beginning of this chapter: the lack of a clear task in single-player games, the number of different abilities that interplay in a campaign and, ultimately, the unreliability of the skill measures.
Figure 4.36: How much each participant overestimated or underestimated the time they had been playing. “Flow error” is the reported time, given in Figure 4.35, minus the actual game time. A positive number indicates an overestimation, and a negative number indicates underestimation, or stronger flow.

The first results that this research chapter presents are the skill measures used in a single-player campaign. Unlike a multiplayer game, where all players compete to accomplish a given task, and can therefore be ranked by their success, the performance measures commonly used are dependent on a multitude of factors, including the game’s difficulty (Figure 4.14). While self-reported measures were found to be more robust, there are still known issues with their reliability, including the Dunning-Kruger effect [10].

While there were a large number of players in the initial data set (187 used in these experiments), only 91 completed the supplied questionnaire. In addition, no participants reported never having played an FPS before. One possibility is that players in the first experiment were primed by their experience playing Project: Blue Room.

Also presented in this chapter was a novel method for measuring difficulty, describing the difficulty of some content as the proportion of completed attempts over some difficulty value, \( d \). This method is independent both of the type of content or game, and the skill measure used. It was then demonstrated that this method produces reasonable difficulty values for different content, automatically adjusting the difficulty value where content is harder. This also means that the game can be described in terms of the player’s skill and the challenge they will
face, two concepts that the player is more likely to understand than the arbitrary
difficulty values assigned by the developer.

A player’s self-reported expertise, $e$, was predicted with a correlation of
$|\rho| = 0.431$ within 30 s of the player playing. This is not as impressive as that
found for multiplayer games, of $|\rho| = 0.874$. However, given an average comple-
tion time was 44 minutes, this is remarkably fast. Other methods for measuring
skill could be used to augment this through the game. The hypothesis presented,
Hypothesis 3, which compared prediction of skill in a single-player game to ran-
domly guessing, can still be accepted, given that it is significantly better than
random guessing ($\rho = 0.0$).

Finally, a method for automatic difficulty selection was presented that em-
ployed all of the above techniques, and compared to a more traditional method
for selecting difficulty: using the player’s self-assessment. The presented model
assigns increasingly challenging difficulties for more skilled players.

To test this method of difficulty selection (ADS), the two groups were com-
pared using four different methods of measuring challenge. The first, Hypoth-
esis 4, compared the similarities in the performance measurements of players.
If performance was drastically different, it indicated players were not being as-
signed similar levels of challenge. The second two, Hypothesis 5 and Hypothesis 6,
compared the reported difficulty and fun of the players, respectively. Finally, Hy-
pothesis 7 stated that players experienced a similar amount of flow with both
systems, using the reported passage of time to test this.

All four of these hypotheses found no significant differences between the two
groups (ADS and self-assessment based). Although players in the control group
reported a slightly lower playing time compared to their actual playing time, indi-
cating more immersion, it was not statistically significant, allowing this research
to accept Hypothesis 7.

For all of the presented hypotheses, the predicted skill was as good as partic-
ipants at selecting a suitable difficulty. In the case of Hypothesis 6, a difficulty
assigned automatically after only 30 s can elicit as much fun from players as using
a difficulty based on self-assessments. However, participants in the experiment
group were assigned much easier difficulties, despite similarities in reported skill
and average progress per second. In order to account for this, it would be possible
to fine-tune the mapping of predicted skill to difficulty without needing to change
predictive models.
This was the first time that skill prediction had been attempted in the context of a single-player game. The constructed model was reasonably adept at predicting player skill, evidenced from the moderate correlation with subjects’ self-assessments ($|\rho| = 0.431$). More importantly, the model’s predictions provided a statistically indistinguishable amount of enjoyment for players compared to difficulties indirectly chosen by themselves. This suggests that even this simple model can be applied in commercial games in place, or in addition to, existing systems, and paves the way for future work, discussed in Chapter 5.
Chapter 5

Conclusions

This research has explored skill capture in the FPS genre of computer games. In this chapter, we reflect on our findings. We first summarize the hypotheses used in this work, then summarize and highlight the main contributions. We then discuss the results and research, including the implications they may have for our understanding of skill capture techniques. Limitations and shortcomings of the research undertaken are also considered as part of this discussion. Finally, we draw these findings together as an impetus for future work, and finish with some overall concluding remarks.

5.1 Summary of results

There were a total of seven hypotheses that were explored in this research, two explored in Chapter 3 and five in Chapter 4. Here we present a summary of each hypothesis and state whether we were able to accept it.

We first tested the feasibility of skill prediction with Hypothesis 1 in a multiplayer game using keyboard and mouse input. This research found in Section 3.5.2 that a predicted skill measure was more reliable than a performance measure taken over the same period of time.

The second hypothesis, Hypothesis 2, deviated from the subject of skill and examined whether it was possible to predict players’ affective states from the same data. Unfortunately, this research was unable to find sufficient evidence to support this hypothesis.

Hypothesis 3 expanded the research into single-player games, and stated that a player’s skill could be predicted in 30 s better than random guessing. With a
Spearman’s \( \rho = 0.431 \) to self-reported measures of skill, we accepted this hypothesis in Section 4.4.2.

The final four hypotheses tested the viability of using skill predictions for selecting appropriate difficulties for the player. The results for these have been presented in Section 4.5.3. Hypothesis 4 first performs this objectively by comparing the similarities of the performances of the groups. Ideally, all players should achieve a similar level of challenge. The assigned difficulties were not statistically different between the control and experiment group, allowing us to accept this hypothesis.

We found similar results with the second and third of these hypotheses, Hypothesis 5 and Hypothesis 6. There was no statistical difference in either the reported difficulty or the reported fun for our method of automatic difficulty selection against a difficulty assigned using the player’s reported skill.

We finally measured and compared the amount of flow players reported between the two groups and no statistical difference was found, allowing us to accept Hypothesis 7.

While we were able to accept the hypotheses presented in this research, the number of participants for each of these experiments was relatively small (20 in each group). We therefore recommend further research in order to assure the validity of this method, in particular determining whether one of the two methods produces more reliable results.

5.2 Discussion

The research has been split into two separate aspects of skill capture: multiplayer games and single-player games. While there is some overlap between the two areas of study, such as the use of input and similar game mechanics, there are large differences in the way people play these games. Reflecting this natural division, the discussions for these are presented separately.

5.2.1 Multiplayer

The primary goal of this research was to demonstrate the feasibility of skill prediction in a first-person shooter. As a side effect of this, however, we presented a data set for this purpose, analysed a variety of skill measures and features and
finally present a model capable of confidently predicting player skill measures in a very short space of time.

The data set

From the FPS *Red Eclipse*, a medium-sized data set was constructed and has been published online [14]. Each game in this data set includes a full log of the player’s input to the game, both through the mouse and the keyboard. The only time when keyboard and mouse data was not recorded was when the player was waiting to respawn. In addition, a number of higher level game events, such as kills and shots fired, were recorded. While the data is not extensively large, containing data from thousands of games, there are additional player responses and demographics recorded with the data, which is more difficult to obtain from online studies.

The players that took part in the experiment reported having played a fairly wide range of experience, according to Figure 3.3. There appears to be a relatively even split between relatively new players, who had played less than 5 FPSs before, and more experienced players. There was also a fair representation of different skills between the two extreme skills in the study, as shown in Figure 3.9. One thing that the data set is missing, however, is professional level players, who may interact differently with the game.

Each game lasted exactly 3 minutes, long enough to measure the player’s skill, and a reasonable length for an in-house experiment. However, real-world games may take longer. *Counter-Strike: Source*, for instance, has a default round limit of 5 minutes [90]. Another disadvantage of a pre-defined time limit is that the games are not necessarily representative of a real-world environment, where games can last an indeterminate amount of time. Rather than playing to a time limit, for instance, a deathmatch can be set to finish when a player reaches a particular kill or score limit.

The most important shortcoming of the data set presented is that players never competed against other players. Instead, the multiplayer aspect was simulated by matching participants against bots. A player’s behaviour may change when knowingly competing against other human players, or even when playing in the same room, when a social aspect is present [102]. Testing this hypothesis might make a fascinating further study, perhaps even a PhD in its own right, and we return to elaborate on how it might be achieved in Section 5.4.
Measuring skill

Once the data had been collected, a variety of skill measures were used, both in-game performance measures and player-reported measures taken from questionnaires. This is, to our knowledge, the first time that different skill measures have been compared in video games in order to examine their validity in the context of the task given to players. Undergoing this process has given extra meaning to the skill prediction, demonstrating that it is indeed skill being predicted, rather than some aspect of the player’s behaviour.

One of the measures explored was TrueSkill, adapted for use against bots. TrueSkill has shown to be meaningful for players in a multiplayer game [48], where players compete against other players. However, for bots, whose skill is defined by the game and therefore fixed, this has not been studied. The method presented here held its validity when compared to other methods, such as the player’s mean score, and it would be interesting to examine its durability when players do not experience similar types of content (as in this case).

Although measures such as player rank, $\bar{r}$, and player score, $\bar{s}$, were shown to be valid, this was only in relation to the given task, a deathmatch. If the game settings were changed, for instance playing to a score limit rather than a time limit, or the game mode was changed, the task would have changed, and new methods for measuring skill must be analysed. However, these measures may be valuable as a proxy for skill in these circumstances.

In addition to the highlighted measures that were shown to be valid ($\bar{r}$, $\bar{s}$ and $T$), other measures were demonstrated to not be, such as average accuracy, $\bar{a}$, and reported hours played per week, $h$. Both of these measures correlated with skill in some respect; players with low accuracy generally performed poorly and players who reported playing at least 10 hours a week performed well. These measures may better describe player behaviour or preferences in the game, or even current playing habits in general, in the case of $h$. However, they should not be used for comparing skill between players for this task.

Feature analysis

A total of 174 features were extracted from each game and categorised according to the type of input, how they were extracted or the amount of knowledge of the game. These groupings allowed us to understand the features from a higher level, rather than on a feature-by-feature basis. In particular, from this research,
the keyboard was found to be the most effective input device, from which the majority of useful features were extracted.

Although the features extracted from the mouse were remarkably poorly correlated to skill, this could be down to the chosen features. However, given their success in previous research [53], this implies that the use of the mouse in an FPS is noisier than in user interfaces. This does not rule out the possibility that there exist suitable features that can be extracted from the movement of the mouse, although this may require observation of different players’ use of the mouse through a game.

Another notable result from the feature analysis was the identification of features related to skill that did not require prior knowledge of the game. Given further research, and identification of features that are more generically applicable to the FPS genre, these features could be used to measure player skill externally from the game. Given the success of keyboard features, this may have security implications where key loggers can be employed.

One notable absence from the list of features extract is that of $n$-grams. An $n$-gram is a specific sequence of items within a longer string of information. While we extracted information about single key presses and some basic information about repetition of key presses through complexity analysis, there may be useful information hidden in the repetition of particular sequences of keys. Skilled players, for instance, may be more inclined to repeat particular sequences of keys, having ingrained patterns. Examining sequences of two or three in length may be trivial, but as the length, $n$, increases, the number of features extracted increases exponentially.

A short analysis of the changes in performance and feature values over the players’ games was also done. In the former case, performance tended to increase as players’ comfort with the game increased. The feature values, on the other hand, as shown in Figure 3.23, converges to its average value much faster. If the player’s skill at controlling the avatar (their mechanical dexterity, introduced in Section 2.1.4) is described by the feature values, an increase in average performance may imply the existence of another aspect of skill, such as prior knowledge, which would need to be acquired over the players’ first few games.
Skill prediction

The most impressive result in this chapter was predicting skill with a high degree of accuracy ($|\rho| = 0.874$). Moreover, this was shown to converge to a reasonable value within 30 s of the game (which is, to our knowledge, the shortest time frame for reasonably accurate assessment of skill in currently published models of skill prediction in video games). This has major implications in the domain of skill capture, where the best skill measures require a minimum of 3 complete games for reliable values [50].

Such an accurate prediction in such a short space of time has applications both in multiplayer games and single-player games, as demonstrated in Chapter 4. One suggested application is for matchmaking, where players can be matched more suitably after a single game, regardless of content, rather than playing multiple games against several opponents. Players could even be asked to play a short game against simulated opponents before being matched. We believe, in any event, that this is one of the central contributions of the research undertaken, and has an exciting potential use in the gaming industry.

Moreover, we believe that Novice Detection, that is detecting which players have no experience of the game and thus need to be guided, is another potential application of the rapid detection methods developed here. Team Fortress 2, for instance, added functionality in 2011 that facilitated coaching of other players. Such a system could employ a novice detection model to suggest coaching for particular players.

The data set was thorough enough to measure skill for each individual and to extract features that could predict each individual’s skill. However, there are only 37 players that were used in the final predictive model, relatively few in terms of gaming. Source games, for instance, support 32 players per game [90, 81]. It would therefore be interesting to see how this predictive model copes with other players.

The predictive models presented in this research were not perfect, still having a correlation to original values less than 0.9. This could be put down to the quality of features extracted, or to the differences in individuals’ skill over the experiment. That the experiment was divided into two phases meant that some players reported having significantly different experience with first-person shooters between phases. The subsequent unreliability of these measures makes it more difficult to predict skill.
The final research presented involved prediction of affective states using the same features. Players who are frustrated may play differently than when they are enjoying themselves. This has been shown to be reflected in higher-level game features [58], but could not be replicated using the player’s input. Although Hypothesis 2 had to be rejected, this does not mean that it cannot be done with the right features and a clear distinction of affective states between players.

### 5.2.2 Single-player

Once skill had been successfully predicted in an FPS, we turned our attention to demonstrating its feasibility in a single-player game, notably more difficult, given the absence of rankings, and the wider array of challenges that the player faced. To accomplish this, we constructed a test-bed, *Project:Blue Room*, from which two sets of data were collected. Using these, we presented an analysis of skill, difficulty, a predictive model, and, finally, an application of skill prediction. These are each described below.

*Project:Blue Room*

For this purpose, a mod of the commercial game *Half-Life 2* was created, named *Project:Blue Room*. The advantage of *Project:Blue Room* as a test-bed over other single-player campaigns was the presence of numerous tropes within a game that could be completed within a manageable time frame (40 minutes). In this respect, it was representative of the genre. It even received several complimentary reviews from the community.\(^1\)

The main issue with the game was the level of challenge for novice players. The game’s difficulty was designed to be as wide enough for the hardest difficulty to be challenging to the best player and the easiest difficulty to be straightforward for the most novice players. However, several players who had never played an FPS before struggled with the puzzles in the second level, Introduction. This was mainly down to the control scheme, and the difficulty navigating non-stationary obstacles in the *Half-Life 2* game. Notably, many of these players reported having some experience in first-person shooters (\(e > 0\)).

In order to simplify the puzzles somewhat, they were simplified for lower difficulties by adding extra boxes. However, some players reported finding it

\(^1\)At the time of writing, the mod has a 7.7 rating from 7 reviews, found on the independently run website, Mod DB: [http://www.moddb.com/mods/projectblue-room](http://www.moddb.com/mods/projectblue-room).
confusing to be given more boxes than were necessary to solve the puzzle.

**The data set**

In the first of the two experiments presented in Chapter 4, 36 participants took part in-house, and 151 online. In the second, there were 40 participants, giving a total of 227 players with a decent proportion of gameplay present. For these players there are 189 hours of gameplay and over 2.75 million events. For many players this data is also accompanied by responses to an expertise questionnaire, giving the players’ own self-assessments of their skill and gaming history.

While this is a significant amount of data and numerous players, only a portion of them completed questionnaires (64 in-house and 67 online). This was enough for the purposes presented in this research, but leaves a significant number of games unlabelled with regards to self-reported measures. Two other issues with these questionnaire responses are that the response of online participants are not as reliable as others, and that the questionnaires were asked of participants after they had taken part in the first experiment.

The data sets used in these experiments again primarily consisted of player input events; 2.19 million events were key presses, in contrast to 556,000 game events. The presence of higher-level game events were, however, more prominent, including more interactions of the player with the game and more information per event. A notable addition was including the player’s position with each event, allowing the player’s progress to be mapped over time. One shortcoming of the events used was the lack of mouse events, which could not feasibly be collected online.

**Skill measures**

Despite having no specific task for which performance could be measured, four skill measures were explored in this research that were directly relevant to single-player games. There were no conclusive results that were found from this brief analysis. The skill measures used could only partially explain the performance differences between players. This was largely down to the difference in difficulties given to players. Unlike in Chapter 3, where participants experienced a variety of contents and difficulties, players of *Project:Blue Room* were given a single, static difficulty.
Another problem with measuring skill in a single-player game is the number of different tasks asked of the player. Players are, for instance, required to complete puzzles, fight enemies and think tactically in places. Any one measure of skill may adequately describe any single aspect of skill, but fall apart when required to differentiate players over several tasks. In order to account for this, skill should be measured separately from the game, e.g. using a short multiplayer game.

This research also presents an updated version of the Game Expertise Questionnaire with questions adapted to better represent skill in a single genre. The number of first-person shooters played, \( f \), presented in both Chapter 3 and Chapter 4, seems a viable method for measuring the player’s skill with a somewhat objective measure. According to Figure 4.11, however, the question does not adequately differentiate between more experienced players. Further research may consider adding a sixth category for 20 games or more.

### Measuring difficulty

Previously, difficulty has been measured using some arbitrary value, such as \( d \) in *Project:Blue Room*, or the bot difficulties in *Red Eclipse*. Instead, we suggest using a method for measuring difficulty that is both independent of the content and the skill measure used, that can also be adapted to use any difficulty value. No feedback is required from the player. Rather, difficulty is assigned relative to the performance of other players.

One shortcoming that was not addressed in this research was that scaling difficulties up is not possible with the current use of ‘probability of completion’. Using some other method for mapping skill to difficulty value from the distribution would solve this issue, however. Additionally, a major obstacle to this method is requiring large amounts of data before the distributions become meaningful.

To our knowledge, this is the first attempt in a video game to measure difficulty in relation to the players and to acquire a difficulty value directly from the skill measurement of the player.

### Skill prediction

Prediction in the single-player game, *Project:Blue Room*, was not as reliable as using a multiplayer game. Correlation of the model’s predicted values of player’s
reported expertise, $\hat{e}$, to the original values, $e$, was only $|\rho| = 0.431$. However, despite an only moderate correlation, we were still able to predict skill significantly better than randomly guessing, and therefore present the first successful attempt at predicting skill within a short space of time in a single-player game.

The suboptimal performance of this model is largely down to the unclear self-reported measures used. Reported expertise has already been shown to be an inaccurate measure of skill [10]. Given performance could not be averaged over different difficulties, it was the best measure at our disposal. Use of more valid measure of skill may improve the accuracy significantly.

Further problems faced in this research are down to the numerous tasks presented to the player in the game. This prediction of the player’s skill is done within the first 30 s of the player beginning the game, which, for all but 5 players, was less than 50 % through the first level. In this time, players are required to perform very little; move around, solve a simple puzzle and, in some cases, break some crates with the crowbar. In contrast, players in Red Eclipse are immediately thrown into the game and must react to the threats presented. This lack of pressure on players in Project:Blue Room may be the cause for the model’s lower performance. Other levels, more representative of the tasks asked of the player, may better reflect the player’s skill.

The features that the model used for prediction that were extracted from game events were either progress-based or weapon-based, two types of events that only occur once the player has completed the first puzzle. In other words, the features used are either input-based or based on how fast the player completed the first puzzle. Backed up by a comparison of the features for different skill groups in Figure 4.24, this provides further evidence that the players’ skills are separated into different aspects, in this case mechanical dexterity and problem solving, that increase at different rates. This is discussed in more depth in Section 5.2.3.

**Automatic difficulty selection**

In a final experiment, we present a method for automatically selecting a difficulty for players (ADS) and compare it to the more traditional method of difficulty selection, using the player’s reported skill. Using the reported expertise as a comparison, ADS assigned reasonable difficulties. Moreover, players did not report a significant difference in difficulty or enjoyment between the two models.

Although the results show that our approach is promising (players whose skill
was predicted enjoyed the game as much as those whose skill was self-reported),
the model prediction accuracy is not as strong as in the multiplayer case. Improv-
ing the predictive power of this model would bring ADS to match the traditional
method of difficulty selection. It seems particularly challenging for the model
to distinguish between the more average players, where \(2 \leq e \leq 4\), as seen in
Figure 4.31.

Another issue is with the experiment itself, in that there were only 20 partici-
pants in each group. Large enough to notice any difference between the groups, it
is not a large enough sample to understand where there are differences or why. Im-
provements on this experiment may include using a third group who were assigned
random difficulties, thereby demonstrating the performance of both methods of
difficulty selection against some basic standard.

In the experiment in Chapter 4, players are asked to report how long they
felt they were playing for. This was done in order to reduce bias from subjective
questionnaires. To our knowledge, this method of inferring flow from reported
time has not yet been used previously. It is therefore possible that this method
suffers from alternative biases, such as players overcompensating for their poor
sense of time. Alternative methods for measuring flow have also been suggested,
that employ more extensive questionnaires for both momentary and complete
measures of flow [103].

ADS has a distinct advantage over traditional difficulty selection in that no
feedback is required of the player. The player is therefore no longer required to
know how good they are and what each difficulty value means. Instead, this is left
to the developer, allowing them to present an appropriate level of challenge. In
terms of Figure 2.2, this allows the developer to assign difficulty in terms of
challenge, even asking the player to select the degree of challenge.

Although the performance of the model did not outperform traditional diffi-
culty selection, ADS provides a reasonable alternative and the potential to out-
perform it after improvements to the predictive model.

5.2.3 A psychological view on Conroy’s framework for
skill

Some of the difficulties encountered in the presented research could be explained
by breaking down skill into component parts. Unfortunately, there is only one
known author that presents such a model in the context of video games [22]. For other purposes, such as education [21], models of skill are more prevalent. In this section, we aim to relate Conroy’s model to the action-perception cycle in cognitive science, which allows a more detailed breakdown, as well as a connection to existing psychological research on skill acquisition.

There is a long history of memory research in psychology for facilitating skilful behaviour [104]. Within this, the action-perception model [105], more recently extended to account for conscious cognition [106], describes how sensory information is processed in the human brain to perform some response, or action. Although there are some variations on terminology and the processes suggested to be involved [107], a simplification has been presented in Figure 5.1. This process is briefly summarised below.

The player initially observes the screen (sensory memory) and the information is passed to the perceptual memory. This is the portion responsible for recognising objects and events, for instance an enemy or a weapon. Recognised objects are transferred to working memory and invoke associations from previous experience (episodic), knowledge of the surroundings (spatial) and factual knowledge (declarative). These memories that feed into the working memory are referred to here as the explicit long-term memory. The process of attention then determines whether the objects are important, passing it into the consciousness where necessary [106]. Once there, the appropriate action can be decided by searching
5.2. DISCUSSION

the procedural memory, essentially a library of known ‘solutions’ (behaviours) applicable in this situation. Once an appropriate algorithm is found it can be passed to the sensory motor memory to be performed.

Conroy defined **prior knowledge** as the player’s long-term and short-term memory of the game. In the context of the action-perception cycle, this could be analogous to the explicit long-term memory that is available to the player on a conscious level. Examples of these are recognition of characters in the game (perceptual), memory of the player’s location (spatial) and a memory of what may have happened the last instance the character was encountered (episodic).

Once these memories have been recalled in the context of some object, it is the attentional processes’ task to bring it to the awareness of the player. If the player has recognised several enemies, for instance, and has recalled the appropriate memories, the ability to identify the correct opponents as important falls under threat detection, as originally defined by Conroy, or, more generally, **situational awareness**. This relies on the contents of the working memory as well as an attentional process bringing this important percept to consciousness.

Before an appropriate action can be taken on the identified objects, some response must be formulated. In the action-perception cycle, possible behavioural responses are stored in procedural memory, and form a subset of what Conroy calls **tactical thinking**. In the context of a single-player game, this may extend to problem solving, or the player’s ability to use the working memory to formulate and test potential strategies.

Finally, once appropriate strategies have been devised, the player can make use of their input, their **mechanical dexterity**, to execute it (motor skills, from a psychological perspective). We argue that the final term coined by Conroy, multi-tasking, is a side-effect of an increase in the other areas. Rather than performing separate actions simultaneously, the player is instead executing them in rapid succession [108], aided by speedier recall, awareness and decision making.

These mappings of Conroy’s terms are not precise. When first beginning to control the avatar, as in mechanical dexterity, the player must recall declarative and episodic knowledge on keys, and situational awareness relies heavily on the player’s explicit long-term memory. However, this generalisation provides some foundation for breaking down skill in a video game.

The implications of an accurate model of distinct components are primarily that skill can be subsequently measured separately. Such a multi-dimensional
model of skill would better describe how players’ skills change between games. In addition, the existing body of research in psychology can provide insight into these components, both in measuring skill and in learning.

Some of the work presented provided evidence for such differences in skill. Novice players, for instance, were easily distinguished by their poor mechanical dexterity, while more advance players were distinguished by the speed of completion of the given task.

5.2.4 Potential application

Having discussed the contributions and the limitations of this work, this section seeks to highlight how the taxonomy and models can be applied in an industry setting. Some of this has been briefly mentioned in previous chapters, but is consolidated here. In order to present this, we suggest a hypothetical racing game to which the various aspects of this thesis can be applied.

Before the data can be collected, the game needs to be instrumented. In order to achieve this, appropriate elements of the game need to be identified, which may serve to indicate a player’s skill. Feature selection techniques will be employed later to identify the most significant features, so an exhaustive list is sufficient here.

The types of features used can be separated into two types: low-level and high-level. This distinction is commonly made in human-computer interaction (HCI) to determine between high and low-level events [109]. Low-level events describe interactions that happen directly between the player and the game, such a key presses or mouse movement. In contrast, high-level events are more abstract, and consist of multiple low-level events. In a racing game, pressing left and right keys would be the lowest-level of event, turning the car left and right would be higher-level, and completing a single lap around a track would be a very high-level.

Low-level events have the potential to be ported to different games or genres, because there will be common actions, such as clicking, that appear in multiple genres. A racing game that uses a keyboard, for instance, would still require the player to press keys in a similar way to a first-person shooter. High-level events, on the other hand, carry more information about the game, and may therefore be more informative to a predictive model.

Instrumentation of the game is the most feasible for developers, who will have
access to the code base and can begin incorporating instrumentation features from the beginning. Instrumentation of older games or by third-party developers is more difficult, given limited access to the source code. Here, low-level instrumentation could be appended by logging keystrokes. Creating features may, however, be time consuming, particularly if an exhaustive approach is taken.

Once instrumentation is complete, **data collection** should be undertaken. The main objectives of this are to:

- Measure the **performances** for a variety of players over several games
- Collect **feature data** associated with these performances
- Construct a **predictive model** of skill
- Determine the **difficulty** of content.

In order to accomplish this, an appropriate performance measure must be used. The most basic measure is a win-loss-draw statistic. Where available, however, rankings should be used for greater descriptive comparison. In many cases, alternative measures can be used from which the ranking is determined. For racing games, this may be the total time taken to complete the race. The content and opponents must also be considered according to the genre when measuring difficulty and skill. For **Red Eclipse** and **Project: Blue Room**, this was the level and the strength of the NPCs against which the player fought. A racing game, however, should consider the times taken on different tracks and the difficulties of the other drivers competing\(^2\).

From this point, the processes presented in this research can be applied directly. The performances recorded can be averaged to produce skill measures, which can be used with the feature data to construct a predictive model. For difficulty selection such as ADS, the techniques presented in Section 4.3.2 can be employed to map the player’s skill to an appropriate difficulty.

Things that can be considered here are the averaging method used for collating the players’ performances as skill measures and the model used for prediction. The techniques presented here were limited, and further techniques may be more appropriate to different games or genres, such as Bayesian averaging for skill measures and artificial neural networks for skill prediction.

\(^2\)For a single player game, the difficulties of the NPCs can be used as in Chapter 4. In a single-player game, the skills of the opponents should be measured.
Data collection may be the most economically challenging process presented here. Developers often employ beta testers to ensure the game is of good quality. While they could be used for data collection, beta testers are usually biased, given they are continuously exposed to all iterations of a game. Similarly, they may not accurately represent the skill of the player base. Moreover, releasing the game before adequate skill adjustment has been done may make the game unplayable for many, damaging the reputation of the game and causing sales to suffer.

Developers may make use of the beta testers to create a preliminary, playable game then fine-tune the predictive models after release. If the difficulties are appropriate on release, players would only benefit from a system that improves in difficulty selection over time.

5.3 Research output

There were several main contributions presented in this research. A brief summary of these is listed here, followed by a more in-depth analysis of them.

- Successful skill prediction in both multiplayer and single-player first-person shooters
- Skill prediction within 30 s of beginning a new game
- A new technique, automatic difficulty selection (ADS), for assigning skill to players
- A medium sized data set of player input in a multiplayer game
- A large data set of player input in a single-player game from both in-house and online participants
- A novel analysis of skill measures and game features with respect to skill
- A new method for measuring difficulty

The most important result presented here is that of skill prediction. Skill was successfully predicted in both multiplayer and single-player first-person shooters. In a multiplayer game, this was done with a correlation of $0.874$ to the original rankings of players (see Section 3.5.2). Moreover, this was feasible within the
first 30 s of beginning a game. These were both significant improvements over the current reported attempts at skill prediction in video games [13, 40, 46].

Difficulty selection traditionally puts the onus on the player to select an appropriate level of difficulty. While some attempts have been made to make this easier for the player using difficulty recommendation [37], ADS is, to our knowledge, the first system to use skill prediction in this way. In the single-player game, Project:Blue Room, this was successfully used to assign difficulties to players who reported no less fun, on average, than those who were assigned difficulties from their own self-assessments.

Two data sets were presented [14], the first of their kind in the FPS genre, and adding to the increasingly wide array of data available for video games [62, 65]. Both data sets included an extensive log of gameplay for a medium-sized cohort of players. These are suitable for exploration of skill in games and for other purposes, as many games are annotated with player feedback and demographics.

A novel analysis of skill measures has been undertaken, analysing the measures in terms of their validity in relation to the task at hand, and their usefulness in skill prediction. Alongside this is a thorough analysis of gameplay features with respect to skill in both a multiplayer and single-player game. Both of these provide a foundation for further research in skill capture.

In games, difficulties are typically represented with a single value [5, 6]. To our knowledge, there is no existing method for measuring difficulty that is independent of the game’s difficulty value and skill measure. Here, one such system was presented, demonstrating that difficulty values could be assigned to players using an arbitrary skill measure.

A final novel contribution of this research was the application of skill prediction techniques to a single-player game. This was demonstrably better than randomly guessing the skill of players (|\rho| = 0.431) with the potential for higher accuracy given measures of skill with higher validity.

Skill capture is typically applied to matchmaking [79, 48], where players are matched against each other based on their skill. Skill prediction, which can be done in a single game, would speed up this process, reducing frustration felt by players when improperly matched. Skill prediction could equally be applied to DDA systems in the context of a single-player game, which traditionally use performance to adapt the difficulty. Furthermore, DDA systems traditionally rely on performance measures, which are noisy and easily manipulated. Using skill
would prevent players feeling cheated when they perform poorly. We argue that the skill capture techniques presented here and, in particular, the skill prediction, would make this process faster and more reliable.

Other potential applications of this work include novice detection. Detecting when players are unfamiliar with the game could allow a tutorial to be presented. Under a system of self-assessment, novices, who overestimated their skill, may miss out, and experts, who could potentially choose to play at a lower difficulty, would be forced to endure it. Skill measures, on the other hand, take too long to determine the player’s skill and, as such, become useless for such a purpose. In contrast, our model detected novice players with 94.9% accuracy in a multiplayer game within the first 30 s of play.

The presented methods of skill capture can also improve on current data collection techniques for developers. Often, demographics of the player are required in order to tailor the game or improve future releases. The information gleaned with these methods could augment existing data.

A final contribution offered is a potential model of skill in video games. There has been no other attempt found in the literature that deconstructs skill into components useful in the context of a video game. Such a model may allow skill to be predicted in a multi-dimensional fashion, potentially tailoring difficulty on a task-by-task basis. This model has therefore been compared to a well-tested model constructed in the domain of psychology and subsequently adapted in order to provide some validity, but will likely require further modifications and rigorous testing in order to be consolidated. In the words of Tulving:

\begin{quote}
In science, as in chess, a plan or a theory, even a poor one, is better than no plan or theory at all. The confusion that usually prevails in the absence of a theory is likely to breed only more of the same, whereas an incorrect theory can always be corrected [104].
\end{quote}

\section{Future work}

Research in skill capture is a relatively new field, with few reliable methods for measuring skill in games [48], and fewer for predicting it [13, 40, 52]. This means that there are many avenues of research left open to explore.

There are a number of shortcomings in this research, as presented in the
preceding section. Most notably is the lack of accurate measures of skill in single-player games. By using alternative ways of measuring skill, these can be compared in the same light as the multiplayer measures have been, exploring the validity of each measure in reference to the tasks assigned to the player.

One alternative method for measuring skill in a single-player game is measuring player performance for individual tasks and either using Bayesian updating over the course of the game, or treating skill as multi-dimensional. A panel of experts could otherwise be used to independently assess the participant’s skill at the game. Regardless of the method, some gold-standard measure of skill must be found in order to continue researching skill in single-player games.

Previous research has shown that a player’s style can be predicted from features of their gameplay [27]. Given the high relevance of our features to skill, further differences may be eliminated by taking into account the player’s style.

This research can also be applied to different game modes, different FPSs and different genres altogether. While applying the findings to similar deathmatch modes may be trivial, predicting skill in different game modes may be trickier, where the player is required to perform different tasks, or even work together as a team. Typical games in *Counter:Strike Source*, for instance, are slower paced, and may require a different model, or even different features. *Team Fortress 2*, on the other hand, is quite fast paced, but allows players to play as different classes, each which has a different style of play. Such differences may require different models, but may equally have some fundamental similarities with regards to player input.

Some more direct research that can be followed on from this is an application to existing data sets. These data sets, such as The Platformer Experience Dataset [62], did not focus on skill, and as such do not have associated records of the player’s reported expertise or any analysis of potential measures of performance. More importantly, fewer recorded the player’s input during the game. However, the relationship between in-game actions and skill could still be explored without the need to collect additional data.

One application of skill prediction has been demonstrated in the context of a single-player game using automatic difficulty selection. Other than direct improvements on the method presented, the model could equally be applied to DDA algorithms, either assigning initial difficulties before the player has even been asked to complete any challenges, or even supplementing the performance heuristics to adapt the difficulty more effectively. Either method could be tested
against the standard DDA algorithm to highlight any improvements.

In a multiplayer context, there is a potential application in matchmaking. The TrueSkill algorithm initially assigns all players the same \( \mu \) and \( \sigma \) values which do not converge to meaningful values for a few games. Instead, players can be assigned more meaningful first estimates of their skill using skill prediction.

The model presented by Conroy was initially designed for constructing AI that better models human behaviour [23]. Should the presented models be shown to adequately predict a player’s mechanical dexterity, this would provide a second building block towards creating human-like opponents.

The experiment conducted in Red Eclipse recorded the entire log of player input, including their mouse movement. We did not find any significant correlations between the player’s skill and the features extracted from their mouse input, but that does not mean there do not exist such features. This means there is still potential in the existing data set. Further work may require in-house observations of the player’s use of the mouse through the game to determine potentially meaningful features.

Players can use vastly different forms of input, such as gamepads, joysticks or even steering wheels. Sometimes this input may map to similar in-game actions, as in the case of gamepads and the keyboard and mouse. Despite this, the results of players can widely differ [110]. Gamepads usually use analogue sticks for both movement and looking, which would require new features and predictive models to be constructed.

In context of multiplayer games, the research presented only explores circumstances where the player competes against simulated opponents. For these purposes, it was assumed that participants would not change their behaviour. However, there is evidence that a social aspect can have a significant impact on the gaming experience [102]. Whether this then impacts the players’ performance or their input is another question that must be answered with respect to the features used in the predictive models and the skill measures. The differences in performance in a real-world environment, for example, may increase as more skilled players take advantage of the less skilled players.

This research presents a method for automatically selecting difficulties, dubbed ADS, which can be added to the existing ensemble of DDA methods that developers have at their disposal [9, 111]. However, the concept of a difficulty dynamically set by the game is, in itself, a contentious issue [112]. Future work should
explore player’s reactions to DDA compared to static difficulties, in particular when the player is told that the system exists. Will players who are assigned a static difficulty and told that it is dynamic still find fault with the supposed DDA algorithm?

Some of the results of this research implied that the models presented were predicting some aspect of skill. In a multiplayer game, this meant that average performance increased where feature values remained similar. In Project:Blue Room, on the other hand, there was a difference between the types of features used for constructing the predictive model.

There is no currently accepted scientific model of skill in video games, that breaks down skill into different aspects. Conroy has suggested such a model that separates skill into four aspects [22], even suggesting that the different aspects have different weights in different games. If skill does consist of separable aspects, then separate models could be constructed that measured or even predicted the different aspects of the player’s skill separately. Not only would this create more reliable models, it would allow the game to change the difficulty of individual challenges.

In order to confirm the existence of these components, the measures of skill must be measured individually, eliminating potential influence of other components. If the resulting measures demonstrate some independence from each other while retaining their validity in the context of the game, this would provide positive evidence for their existence. One example would be to have players comfortable with a keyboard and mouse play the same game with a gamepad and vice versa, demonstrating the effect of the input, or the player’s mechanical dexterity. Alternatively, players could be asked to complete distinct tasks, each designed to test a different component.

The assumptions of skill in this research were that it was a single-dimension and was stable over time. In the games presented here, for instance, the skill was measured over several games or levels. However, if skill indicates the player’s probability of winning a particular game, it is likely dependent on numerous factors external to the game, including the player’s current health, their environment, or even the day of the week. If the effects of any external factors on skill can be determined, a more representative measure of skill can be taken, that is resistant to these effects.

Some of this research is based on currently unchallenged assumptions about
player preference. In particular, the degree of challenge that players prefer to face. While previous research has demonstrated bias in skill assessment [10], it is currently unknown how players select difficulty, both in relation to their own skill and in relation to the degree of challenge they wish to experience. This research question also applies to matchmaking, where it is assumed that the optimal experience is one in which a draw is likely. It may be that this is insufficient for modelling player satisfaction or learning in games, and some combination of challenges over time is required.

More generally, skill capture applies to a wide variety of applications. Where any form of learning is required, it is important to be able to measure people’s skill in order to assign the appropriate challenges to optimise learning. It is important in language learning, for instance, to present the student with the correct amount of vocabulary given their current grasp of the language. Traditionally, the teacher makes these assessments as the student progresses, but as more tools become automated and online, it becomes increasingly important to assess their skill automatically.

5.5 Concluding remarks

We have examined methods of skill capture in two first-person shooters in both a multiplayer and a single-player context, presenting a framework for automatically predicting skill in a short amount of time (Chapter 3) and for using such evaluations for automatically selecting game difficulty (Chapter 4). We have substantiated the viability of the former approach by obtaining strong correlations with player score within 30 s of play, and the latter by showing that players enjoyed games using automatically selected difficulties as much as when they were selected using their own self-assessments.

Our results pave the way for future work in rapid skill capture on a sub-minute time scale, and provide prototypes of tools for future game developers to estimate player skill in a more unbiased fashion than self-assessment and significantly faster than existing methods.
Bibliography


[31] “SteamSaga,” [Unreleased], Quantum Tiger Games.


[38] “God Hand,” [PlayStation 2], Capcom, 2006.


[78] M. B. Brown and A. B. Forsythe, “Robust tests for the equality of vari-
ances,” Journal of the American Statistical Association, vol. 69, no. 346,

[79] O. Delalleau, E. Contal, E. Thibodeau-Laufer, R. Ferrari, Y. Bengio, and
F. Zhang, “Beyond skill rating: Advanced matchmaking in Ghost Recon
online,” IEEE Trans. Computational Intell. and AI Games, vol. 4, no. 3,

http://sauerbraten.org


[83] D. Kahneman, “Choices, values and frames,” in Experienced Utility and Ob-
jective Happiness: A Moment-Based Approach, D. Kahneman and A. Tver-
673–692.

Proc. International Cross-Disciplinary Conf. Web Accessibility (W4A’13),

[85] G. T. Fechner, Elemente der Psychophysik. Leipzig: Breitkopf & Härtel,
1860.

corpus for analysing the interplay between game context and player experi-
Notes in Computer Science, S. D.Mello, A. Graesser, B. Schuller, and J.-C.

[87] R. Likert, A Technique for the Measurement of Attitudes, ser. Archives of
Psychology. s.n., 1932.

strategy for a two-player video game,” Procedia Technology, vol. 7, no. 0,


Appendix A

The Game Expertise Questionnaire

The Game Expertise Questionnaire has been used successfully in previous research[64]. For this reason, it provides a solid foundation for measuring self-reported skill in players. An adapted form of this questionnaire has been used in Chapter 4, described more thoroughly in Section 4.2.4. The original questionnaire has been included in this appendix for reference.
### History of video game playing survey

**Gender:** M / F

How many years have you been playing videogames (if you play): ______

For each of the following game types, please CIRCLE the appropriate response for:
1. **Your EXPERTISE** level (1-7) for your most extensively played game in that category
2. **Your CURRENT HOURS PER WEEK** on average (for the past 6-months)
3. If you’ve **EVER** played the game type more than 5 hours a week (for example, in high school).

<table>
<thead>
<tr>
<th>EXPERTISE</th>
<th>HRS/WK (last 6mths)</th>
<th>5+hrs/wk EVER?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 3 4 5 6 7</td>
<td>0 1-2 3-6 7-12 13+</td>
</tr>
</tbody>
</table>

#### 1. LOGIC/PUZZLE

Some examples: (circle any played extensively, or write in)
- Solitaire
- Pokemon
- Bejeweled
- The Sims
- Civilization
- Tycoon games

#### 2. DEXTERITY GAMES

Some examples: (circle any played extensively, or write in)
- Tetris
- Bust-a-move
- DS games

#### 3. REAL-TIME STRATEGY

Some examples: (circle any played extensively, or write in)
- Starcraft
- Warcraft
- Age of Empires

#### 4. 1st PERSON SHOOTER

Some examples: (circle any played extensively, or write in)
- Halo
- Gears of War
- Quake
- BioShock
- Unreal Tournament
- Counterstrike
- Resistance:FoM

#### 5. SPORTS/ACTION

Some examples: (circle any played extensively, or write in)
- Madden
- MarioKart
- NCAA
- Burnout
- Gran Turismo
- God of War

#### 6. OTHER GAMES

Some examples: (circle any played extensively, or write in)
- World of Warcraft
- Everquest
- TextTwist
- Guitar Hero
- Dance Dance Revolution
- Diablo

Any other games that you would consider yourself an expert at which didn’t fit any category: __________

---

Figure A.1: The original Game Expertise Questionnaire.
Appendix B

Red Eclipse Features

This appendix includes a summarised list of features extracted from Red Eclipse. The least interesting features have been omitted for the sake of brevity. While also providing a reference for the main research in Chapter 3, this list can provide insight into the most useful features for skill capture, providing a foundation for further research. The least correlated features in this list have been kept in order to avoid their reuse.

The name for each feature is its given name in the associated data set\textsuperscript{1}. A description is also given as a summary for how it was extracted, which of the groups it belongs to for each of the three groupings, and the Pearson’s correlation, $r$, to player score, $\bar{s}$. Strongly correlated features, where $r < -0.6$ and $r > 0.6$ are highlighted.

Some features have been extracted from a path representation of the player’s input. For the mouse, this is the path constructed by joining subsequent mouse positions together. The ‘movement path’ is constructed by tracing a line using the state of the four movement keys, forwards, left, right and back. Holding forwards, for instance, would trace the path in the y axis, and left and right on the x axis. The ‘position path’ used in some features was constructed from a combination of the movement keys and mouse input left and right. This became a proxy for the player’s position in the game.

The first of the three groupings presented here, \textit{hardware}, describes the type of input the feature was extracted from, keyboard, mouse movement or mouse clicks. The second, \textit{type}, is the type of feature that was extracted and \textit{context} indicates whether the features requires prior knowledge of the game to extract.

\footnote{\texttt{https://www.escholar.manchester.ac.uk/uk-ac-man-scw:262244}}
They are each described and analysed in more detailed in Section 3.4.

45 features were extracted using a Fourier transform and describe each of its first 10 frequencies (or 5 in the case of keyboard events per second). These were extracted from the movement path, the mouse and the keyboard events. The majority of these have been omitted due to their low correlation and low relative interest.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Hardware</th>
<th>Type</th>
<th>Context</th>
<th>Pearson's $r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>numdiffkeys</td>
<td>The total number of different keys the player uses.</td>
<td>Keyboard</td>
<td>Free</td>
<td>0.4762</td>
<td></td>
</tr>
<tr>
<td>mean-numkeys</td>
<td>The average number of keys pressed at one time.</td>
<td>Keyboard</td>
<td>Free</td>
<td>0.7896</td>
<td></td>
</tr>
<tr>
<td>max-numkeys</td>
<td>The most number of keys pressed at one time.</td>
<td>Keyboard</td>
<td>Free</td>
<td>0.6365</td>
<td></td>
</tr>
<tr>
<td>zlib-keystr</td>
<td>The zlib complexity of the command string (a representation where commands are represented by single characters, e.g. ws indicates the player pressed forwards followed by back).</td>
<td>Keyboard</td>
<td>Complexity Dependent</td>
<td>-0.4057</td>
<td></td>
</tr>
<tr>
<td>huffman-keystr</td>
<td>The Huffman complexity of the command string.</td>
<td>Keyboard</td>
<td>Complexity Dependent</td>
<td>0.6925</td>
<td></td>
</tr>
<tr>
<td>entropy-keystr</td>
<td>The frequency entropy of the command string.</td>
<td>Keyboard</td>
<td>Complexity Dependent</td>
<td>-0.5447</td>
<td></td>
</tr>
<tr>
<td>samp-keybin</td>
<td>The sample entropy of the keys' binary string (the keys pressed represented by the ASCII code in binary).</td>
<td>Keyboard</td>
<td>Complexity Dependent</td>
<td>0.5801</td>
<td></td>
</tr>
<tr>
<td>zlib-bintostr-keybin</td>
<td>The zlib complexity of the keys' binary string.</td>
<td>Keyboard</td>
<td>Complexity Dependent</td>
<td>0.7578</td>
<td></td>
</tr>
<tr>
<td>huffman-keybin</td>
<td>The Huffman complexity of the keys' binary string.</td>
<td>Keyboard</td>
<td>Complexity Dependent</td>
<td>0.7452</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Hardware</td>
<td>Type</td>
<td>Context</td>
<td>Pearson’s $r$</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------</td>
<td>---------</td>
<td>---------</td>
<td>----------------</td>
</tr>
<tr>
<td>11     lz-bintostr-keybin</td>
<td>The LZW complexity of the keys’ binary string.</td>
<td>Keyboard</td>
<td>Complexity</td>
<td>Dependent</td>
<td>0.7994</td>
</tr>
<tr>
<td>12     lz-inttostr-keybin</td>
<td>The LZW complexity of the keys’ ASCII string.</td>
<td>Keyboard</td>
<td>Complexity</td>
<td>Dependent</td>
<td>0.7579</td>
</tr>
<tr>
<td>13     lz-keystr</td>
<td>The LZW complexity of the commandstring.</td>
<td>Keyboard</td>
<td>Complexity</td>
<td>Dependent</td>
<td>0.5427</td>
</tr>
<tr>
<td>14     lz-wbinary</td>
<td>The LZW complexity of the forward key’s binary string, i.e. for each key event, indicates whether or not the forward key is pressed.</td>
<td>Keyboard</td>
<td>Complexity</td>
<td>Dependent</td>
<td>-0.6258</td>
</tr>
<tr>
<td>15     lz-sbinary</td>
<td>The LZW complexity of the backward key’s binary string.</td>
<td>Keyboard</td>
<td>Complexity</td>
<td>Dependent</td>
<td>0.3300</td>
</tr>
<tr>
<td>16     lz-abinary</td>
<td>The LZW complexity of the left key’s binary string.</td>
<td>Keyboard</td>
<td>Complexity</td>
<td>Dependent</td>
<td>0.3497</td>
</tr>
<tr>
<td>17     lz-dbinary</td>
<td>The LZW complexity of the right key’s binary string.</td>
<td>Keyboard</td>
<td>Complexity</td>
<td>Dependent</td>
<td>0.3225</td>
</tr>
<tr>
<td>18     multikeys/1</td>
<td>The total time for which at least one key is pressed.</td>
<td>Keyboard</td>
<td>Free</td>
<td></td>
<td>0.7441</td>
</tr>
<tr>
<td>19     multikeys/2</td>
<td>The total time for which at least two keys are pressed.</td>
<td>Keyboard</td>
<td>Free</td>
<td></td>
<td>0.7802</td>
</tr>
<tr>
<td>20     multikeys/3</td>
<td>The total time for which at least three keys are pressed.</td>
<td>Keyboard</td>
<td>Free</td>
<td></td>
<td>0.5712</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Hardware</td>
<td>Type</td>
<td>Context</td>
<td>Pearson’s $r$</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
<td>---------------</td>
<td>-----------</td>
<td>---------------</td>
</tr>
<tr>
<td>22 height-movement</td>
<td>The first dimension of curvature on the player’s movement path (a path representation of the player’s key presses forward, left, right and back).</td>
<td>Keyboard</td>
<td>Dependent</td>
<td>0.3244</td>
<td></td>
</tr>
<tr>
<td>23 width-movement</td>
<td>The second dimension of curvature on the player’s movement path.</td>
<td>Keyboard</td>
<td>Dependent</td>
<td>0.4635</td>
<td></td>
</tr>
<tr>
<td>24 cross-movement</td>
<td>The third dimension of curvature on the player’s movement path.</td>
<td>Keyboard</td>
<td>Dependent</td>
<td>0.3452</td>
<td></td>
</tr>
<tr>
<td>25 angles-mean-movement</td>
<td>The average size of the angle between successive vectors in the player’s movement path.</td>
<td>Keyboard</td>
<td>Kinetics</td>
<td>0.4644</td>
<td></td>
</tr>
<tr>
<td>26 angles-std-movement</td>
<td>The standard deviation of the angles between successive vectors in the player’s movement path.</td>
<td>Keyboard</td>
<td>Kinetics</td>
<td>-0.1561</td>
<td></td>
</tr>
<tr>
<td>27 eventfreq-movement</td>
<td>The number of key presses per millisecond.</td>
<td>Keyboard</td>
<td>Event Frequency</td>
<td>-0.2408</td>
<td></td>
</tr>
<tr>
<td>28 occupancy-flatten-mean-movement</td>
<td>The mean of a flattened frequency grid of the player’s movement path (a frequency is the number of times events happen in each area).</td>
<td>Keyboard</td>
<td>Dependent</td>
<td>0.6911</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Hardware</td>
<td>Type</td>
<td>Context</td>
<td>Pearson’s $r$</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------</td>
<td>---------------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>29 occupancy-flatten-std-movement</td>
<td>The standard deviation of a flattened frequency grid of the player’s movement path.</td>
<td>Keyboard</td>
<td>Dependent</td>
<td>0.4510</td>
<td></td>
</tr>
<tr>
<td>32 regressionerror-movement</td>
<td>The mean-squared error for fitting a regression curve of the points in the player’s movement path (how easy it was to fit).</td>
<td>Keyboard</td>
<td>Dependent</td>
<td>0.4054</td>
<td></td>
</tr>
<tr>
<td>35 length-movement</td>
<td>The total length of the player’s movement path.</td>
<td>Keyboard</td>
<td>Kinetics</td>
<td>0.7804</td>
<td></td>
</tr>
<tr>
<td>36 straights-movement</td>
<td>The total length of the straight lines (where the player only pressed one key) in the player’s movement path divided by the total length of the path.</td>
<td>Keyboard</td>
<td>Dependent</td>
<td>-0.7473</td>
<td></td>
</tr>
<tr>
<td>37 intersections-movement</td>
<td>The number of intersections in the player’s movement path.</td>
<td>Keyboard</td>
<td>Dependent</td>
<td>0.6639</td>
<td></td>
</tr>
<tr>
<td>38 x-mean-sample-fourier-movement</td>
<td>The mean of a fourier transform taken over the players left and right strafing.</td>
<td>Keyboard</td>
<td>Complexity</td>
<td>0.3625</td>
<td></td>
</tr>
<tr>
<td>39 y-mean-sample-fourier-movement</td>
<td>The mean of a fourier transform taken over a player’s forwards and back movement.</td>
<td>Keyboard</td>
<td>Complexity</td>
<td>0.3196</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Hardware</td>
<td>Type</td>
<td>Context</td>
<td>Pearson’s r</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
<td>-------------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>forwardtime-movement</td>
<td>The total time the player is moving forwards.</td>
<td>Keyboard</td>
<td>Dependent</td>
<td></td>
<td>0.6355</td>
</tr>
<tr>
<td>backwardtime-movement</td>
<td>The total time the player is moving backwards.</td>
<td>Keyboard</td>
<td>Dependent</td>
<td></td>
<td>0.6312</td>
</tr>
<tr>
<td>lefftime-movement</td>
<td>The total time the player strafes left.</td>
<td>Keyboard</td>
<td>Dependent</td>
<td></td>
<td>0.6706</td>
</tr>
<tr>
<td>righttime-movement</td>
<td>The total time the player strafes right.</td>
<td>Keyboard</td>
<td>Dependent</td>
<td></td>
<td>0.6036</td>
</tr>
<tr>
<td>presses-num-requiredkeys</td>
<td>The number of key events for the most basic set of keys</td>
<td>Keyboard</td>
<td>Event Frequency</td>
<td>Dependent</td>
<td>0.7228</td>
</tr>
<tr>
<td>keytime-requiredkeys</td>
<td>The total time the basic set of keys are pressed.</td>
<td>Keyboard</td>
<td>Event Frequency</td>
<td>Dependent</td>
<td>0.7663</td>
</tr>
<tr>
<td>presses-num-common</td>
<td>The number of key events for a set of commonly pressed keys</td>
<td>Keyboard</td>
<td>Event Frequency</td>
<td>Dependent</td>
<td>0.4075</td>
</tr>
<tr>
<td>sum-magnitude-displacement</td>
<td>The player’s movement during clicking bursts (periods where the player performs lots of clicking).</td>
<td>Clicks</td>
<td>Dependent</td>
<td></td>
<td>0.7738</td>
</tr>
<tr>
<td>clickBursts-movement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>num-clicks/10</td>
<td>The number of clicking bursts.</td>
<td>Clicks</td>
<td>Event Frequency</td>
<td>Free</td>
<td>-0.2602</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Hardware</td>
<td>Type</td>
<td>Context</td>
<td>Pearson’s $r$</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------</td>
<td>---------------</td>
<td>---------</td>
<td>---------------</td>
</tr>
<tr>
<td>mean-bursttime-clicks</td>
<td>The average length of the clicking bursts.</td>
<td>Clicks</td>
<td>Free</td>
<td>0.2692</td>
<td></td>
</tr>
<tr>
<td>lz-clickbin</td>
<td>The LZW complexity of the player’s clicking as a binary string.</td>
<td>Clicks</td>
<td>Complexity</td>
<td>Free</td>
<td>0.4184</td>
</tr>
<tr>
<td>perclicks-ammoevents</td>
<td>The amount of ammo used divided by the number of clicks.</td>
<td>Clicks</td>
<td>Dependent</td>
<td>0.2049</td>
<td></td>
</tr>
<tr>
<td>length-position</td>
<td>The length of the player’s position path.</td>
<td>Kinetics</td>
<td>Dependent</td>
<td>0.7662</td>
<td></td>
</tr>
<tr>
<td>eventfreq-position</td>
<td>The number of input events per milisecond.</td>
<td>Event Frequency</td>
<td>Dependent</td>
<td>0.5948</td>
<td></td>
</tr>
<tr>
<td>occupancy-flatten-mean-position</td>
<td>The mean value of a flattened frequency grid of the player’s position path.</td>
<td>Dependent</td>
<td>0.6222</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ratio-mouse</td>
<td>The ratio of vertical to horizontal mouse movement.</td>
<td>Mouse</td>
<td>Free</td>
<td>0.1080</td>
<td></td>
</tr>
<tr>
<td>width-mouse</td>
<td>The first dimension of curvature on the player’s mouse movement.</td>
<td>Mouse</td>
<td>Free</td>
<td>0.1744</td>
<td></td>
</tr>
<tr>
<td>height-mouse</td>
<td>The second dimension of curvature on the player’s mouse movement.</td>
<td>Mouse</td>
<td>Free</td>
<td>0.0687</td>
<td></td>
</tr>
<tr>
<td>cross-mouse</td>
<td>The third dimension of curvature on the player’s mouse movement.</td>
<td>Mouse</td>
<td>Free</td>
<td>0.1753</td>
<td></td>
</tr>
<tr>
<td>length-mouse</td>
<td>The total distance the mouse moved.</td>
<td>Mouse</td>
<td>Kinetics</td>
<td>Free</td>
<td>0.2620</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Hardware</td>
<td>Type</td>
<td>Context</td>
<td>Pearson’s $r$</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------</td>
<td>----------------</td>
<td>---------</td>
<td>---------------</td>
</tr>
<tr>
<td>eventfreq-mouse</td>
<td>The number of mouse movement events per milisecond.</td>
<td>Mouse</td>
<td>Event Frequency</td>
<td>Free</td>
<td>-0.0189</td>
</tr>
<tr>
<td>magnitude-mean-mouse</td>
<td>The player’s mean position of the mouse.</td>
<td>Mouse</td>
<td>Kinetics</td>
<td>Free</td>
<td>0.1271</td>
</tr>
<tr>
<td>magnitude-vel-mean-mouse</td>
<td>The mean velocity of the mouse.</td>
<td>Mouse</td>
<td>Kinetics</td>
<td>Free</td>
<td>-0.1050</td>
</tr>
<tr>
<td>magnitude-accel-max-mouse</td>
<td>The maximum acceleration of the mouse.</td>
<td>Mouse</td>
<td>Kinetics</td>
<td>Free</td>
<td>-0.0513</td>
</tr>
<tr>
<td>angles-mean-mouse</td>
<td>The average change of angle between successive mouse movements.</td>
<td>Mouse</td>
<td>Kinetics</td>
<td>Free</td>
<td>0.1357</td>
</tr>
<tr>
<td>angles-std-mouse</td>
<td>The standard deviation of the angles between successive mouse movements.</td>
<td>Mouse</td>
<td>Kinetics</td>
<td>Free</td>
<td>0.1380</td>
</tr>
<tr>
<td>right-mouse</td>
<td>The maximum x position of the mouse.</td>
<td>Mouse</td>
<td></td>
<td>Free</td>
<td>0.2104</td>
</tr>
<tr>
<td>left-mouse</td>
<td>The minimum x position of the mouse.</td>
<td>Mouse</td>
<td></td>
<td>Free</td>
<td>0.0777</td>
</tr>
<tr>
<td>x-vel-mean-mouse</td>
<td>The mean velocity of the mouse in the x direction.</td>
<td>Mouse</td>
<td>Kinetics</td>
<td>Free</td>
<td>0.0918</td>
</tr>
<tr>
<td>x-accel-mean-mouse</td>
<td>The mean acceleration of the mouse in the y direction.</td>
<td>Mouse</td>
<td>Kinetics</td>
<td>Free</td>
<td>-0.0241</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Hardware</td>
<td>Type</td>
<td>Context</td>
<td>Pearson’s $r$</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------</td>
<td>----------</td>
<td>---------</td>
<td>---------------</td>
</tr>
<tr>
<td>x-accel-max-mouse</td>
<td>The maximum acceleration of the mouse in the x direction.</td>
<td>Mouse</td>
<td>Kinetics</td>
<td>Free</td>
<td>-0.0769</td>
</tr>
<tr>
<td>xturns-num-mouse</td>
<td>The number of times the mouse changed direction relative to its previous direction.</td>
<td>Mouse</td>
<td>Kinetics</td>
<td>Free</td>
<td>0.4947</td>
</tr>
<tr>
<td>xturns-time-mean-mouse</td>
<td>The average time between the mouse changing direction.</td>
<td>Mouse</td>
<td>Kinetics</td>
<td>Free</td>
<td>-0.1161</td>
</tr>
<tr>
<td>xturns-time-max-mouse</td>
<td>The maximum time between the mouse changing direction.</td>
<td>Mouse</td>
<td>Kinetics</td>
<td>Free</td>
<td>-0.2360</td>
</tr>
<tr>
<td>x-lz-contobin-mouse</td>
<td>The LZW complexity of mouse movement in the x direction.</td>
<td>Mouse</td>
<td>Complexity</td>
<td>Free</td>
<td>-0.3402</td>
</tr>
<tr>
<td>x-zlib-contobin-mouse</td>
<td>The zlib complexity of mouse movement in the x direction.</td>
<td>Mouse</td>
<td>Complexity</td>
<td>Free</td>
<td>-0.1690</td>
</tr>
<tr>
<td>x-std-mouse</td>
<td>The standard deviation of x points in mouse movement events.</td>
<td>Mouse</td>
<td>Free</td>
<td>0.1595</td>
<td></td>
</tr>
<tr>
<td>y-mean-mouse</td>
<td>The mean y position of the mouse.</td>
<td>Mouse</td>
<td>Free</td>
<td>0.1349</td>
<td></td>
</tr>
<tr>
<td>y-vel-mean-mouse</td>
<td>The average velocity of the mouse in the y direction.</td>
<td>Mouse</td>
<td>Kinetics</td>
<td>Free</td>
<td>0.2203</td>
</tr>
<tr>
<td>y-accel-mean-mouse</td>
<td>The average acceleration of the mouse in the y direction.</td>
<td>Mouse</td>
<td>Kinetics</td>
<td>Free</td>
<td>0.2161</td>
</tr>
<tr>
<td>y-accel-max-mouse</td>
<td>The maximum acceleration of the mouse in the y direction.</td>
<td>Mouse</td>
<td>Kinetics</td>
<td>Free</td>
<td>-0.0234</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Hardware</td>
<td>Type</td>
<td>Context</td>
<td>Pearson’s $r$</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
<td>--------------</td>
<td>---------</td>
<td>---------------</td>
</tr>
<tr>
<td>103 y-mean-clickBursts-mouse</td>
<td>The mean maximum y displacement while holding a mouse button.</td>
<td>Mouse</td>
<td></td>
<td>Free</td>
<td>0.1349</td>
</tr>
<tr>
<td>104 y-sample-samp-mouse</td>
<td>The sample entropy for mouse movement in the y direction.</td>
<td>Mouse</td>
<td>Complexity</td>
<td>Free</td>
<td>0.1762</td>
</tr>
<tr>
<td>105 y-lz-contobin-mouse</td>
<td>The LZW complexity of mouse movement in the y direction.</td>
<td>Mouse</td>
<td>Complexity</td>
<td>Free</td>
<td>-0.1301</td>
</tr>
<tr>
<td>106 y-zlib-contobin-mouse</td>
<td>The zlib complexity of mouse movement in the y direction.</td>
<td>Mouse</td>
<td>Complexity</td>
<td>Free</td>
<td>-0.0225</td>
</tr>
<tr>
<td>110 num-mouseevents</td>
<td>The number of mouse movement events.</td>
<td>Mouse</td>
<td>Event Frequency</td>
<td>Free</td>
<td>0.3638</td>
</tr>
<tr>
<td>112 time-mean-mouseevents</td>
<td>The mean time between mouse movement events.</td>
<td>Mouse</td>
<td>Event Frequency</td>
<td>Free</td>
<td>-0.3728</td>
</tr>
<tr>
<td>113 time-std-mouseevents</td>
<td>The standard deviation of time between mouse movement events.</td>
<td>Mouse</td>
<td>Event Frequency</td>
<td>Free</td>
<td>-0.3764</td>
</tr>
<tr>
<td>114 time-max-mouseevents</td>
<td>The maximum time between mouse movement events.</td>
<td>Mouse</td>
<td>Event Frequency</td>
<td>Free</td>
<td>-0.3224</td>
</tr>
<tr>
<td>115 time-min-mouseevents</td>
<td>The minimum time between mouse movement events.</td>
<td>Mouse</td>
<td>Event Frequency</td>
<td>Free</td>
<td>-0.1242</td>
</tr>
<tr>
<td>116 num-clickevents</td>
<td>The number of mouse clicks.</td>
<td>Clicks</td>
<td>Event Frequency</td>
<td>Free</td>
<td>-0.1234</td>
</tr>
<tr>
<td>118 time-mean-clickevents</td>
<td>The mean time between mouse clicks.</td>
<td>Clicks</td>
<td>Event Frequency</td>
<td>Free</td>
<td>0.0964</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Hardware</td>
<td>Type</td>
<td>Context</td>
<td>Pearson’s r</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------</td>
<td>----------</td>
<td>---------------------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>120 time-max-clickevents</td>
<td>The maximum time between mouse clicks.</td>
<td>Clicks</td>
<td>Event Frequency</td>
<td>Free</td>
<td>-0.2577</td>
</tr>
<tr>
<td>121 time-min-clickevents</td>
<td>The minimum time between mouse clicks.</td>
<td>Clicks</td>
<td>Event Frequency</td>
<td>Free</td>
<td>-0.0720</td>
</tr>
<tr>
<td>122 num-keyevents</td>
<td>The number of key press events.</td>
<td>Keyboard</td>
<td>Event Frequency</td>
<td>Free</td>
<td>0.7573</td>
</tr>
<tr>
<td>124 time-mean-keyevents</td>
<td>The mean time between key press events.</td>
<td>Keyboard</td>
<td>Event Frequency</td>
<td>Free</td>
<td>-0.7035</td>
</tr>
<tr>
<td>125 time-std-keyevents</td>
<td>The standard deviation of time between key press events.</td>
<td>Keyboard</td>
<td>Event Frequency</td>
<td>Free</td>
<td>-0.6482</td>
</tr>
<tr>
<td>126 time-max-keyevents</td>
<td>The maximum time between key press events.</td>
<td>Keyboard</td>
<td>Event Frequency</td>
<td>Free</td>
<td>-0.5157</td>
</tr>
<tr>
<td>127 time-min-keyevents</td>
<td>The minimum time between key press events.</td>
<td>Keyboard</td>
<td>Event Frequency</td>
<td>Free</td>
<td>-0.2550</td>
</tr>
<tr>
<td>168 index/0-fourier-eventsample-keyevents</td>
<td>The 1st frequency of a fourier analysis of the number of keyboard events per second.</td>
<td>Keyboard</td>
<td>Complexity</td>
<td>Free</td>
<td>0.7573</td>
</tr>
<tr>
<td>173 mean-eventsample-keyevents</td>
<td>The average number of key events per second.</td>
<td>Keyboard</td>
<td>Event Frequency</td>
<td>Free</td>
<td>0.7573</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Hardware</td>
<td>Type</td>
<td>Context</td>
<td>Pearson’s $r$</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------</td>
<td>-------------------</td>
<td>---------</td>
<td>---------------</td>
</tr>
<tr>
<td>std-eventsample-keyevents</td>
<td>The standard deviation of the number of key events per second.</td>
<td>Keyboard</td>
<td>Event Frequency</td>
<td>Free</td>
<td>0.7061</td>
</tr>
</tbody>
</table>
Appendix C

*Project: Blue Room* Features

This appendix serves to compliment the research presented in Chapter 4. The features are listed in the reverse order that they are removed using the wrapper method. In other words, the first features were deemed the most important, and the features at the end caused the least impact by their removal. This is represented by the number given with each feature.

As in Appendix B, this list of features can serve as an informative guide to extracting features in a single-player FPS, and potentially a basis for work in other genres. For each, the correlation presented, Spearman’s $|\rho|$, indicates the accuracy of the model after the given feature has been removed.

A 'burst', sometimes referred to by features, is a group of events than are no further than 2 s apart. Some features also reference key bindings, such as +forward. These are the commands issued to the game by the keyboard presses or mouse clicks.
| Name                        | Description                                                                 | Spearman’s $|\rho|$ |
|-----------------------------|-----------------------------------------------------------------------------|------------|
| numdiffkeys                 | The total number of different keys the player uses.                         | -          |
| persec-movement             | The total distance moved by the player per second.                          | 0.271      |
| progress-maxtime            | The maximum time between any two Progress events.                           | 0.301      |
| weaponpickup-bursts         | The average number of WeaponPickup bursts per second.                       | 0.284      |
| mean-persec-keyspressed     | The average number of key events per second time window, after grouping     | 0.308      |
| num-crowbar                 | The number of times the crowbar was swung.                                  | 0.334      |
| keyup-lz-bin                | The LZW complexity of KeyUp events.                                         | 0.346      |
| progress-std                | The standard deviation of number of Progress events per second.             | 0.367      |
| weaponpickup-lz-num         | The LZW complexity of WeaponPickup events.                                  | 0.373      |
| shoot-lz-num                | The LZW complexity of Shoot events.                                         | 0.395      |
| all-std                     | The standard deviation of number of events per second, grouping events into second clusters. | 0.411      |
| persec-movementkeys         | The total time the player held the movement keys per second.                | 0.417      |
| keytime-                    | The total time the player held any of the ’required’ keys (\texttt{+forward, requiredkeys} \texttt{+moveleft, +moveright, +back, +attack, +use, +jump and +duck}). | 0.395      |
| shoot-std                   | The standard deviation of Shoot events per second.                          | 0.404      |
| lz-forwards                 | The LZW complexity of the player’s forward key presses (represented as a binary string). | 0.421      |
| freqentropy-command         | The Shannon entropy of the player’s key presses.                           | 0.428      |
| keyup-maxtime               | The maximum time between any two KeyUp events.                              | 0.431      |
| progress-lz-num             | The LZW complexity of the Progress events.                                 | 0.429      |
| Name                  | Description                                                                 | Spearman’s $|\rho|$ |
|-----------------------|------------------------------------------------------------------------------|-----------|
| keydown-maxtime       | The maximum time between any two KeyDown events.                             | 0.426     |
| huffman-keystr        | The Huffman complexity of the player’s input.                               | 0.419     |
| keydown-stdtime       | The standard deviation of the times between KeyDown events.                 | 0.410     |
| keyup-lz-num          | The LZW complexity of the KeyUp events.                                     | 0.409     |
| keydown-lz-bin        | The LZW complexity of the KeyDown events, represented as a binary string.   | 0.411     |
| shoot-bursts          | The number of bursts of Shoot events.                                        | 0.413     |
| weaponselect-num      | The number of WeaponSelect events per second.                               | 0.411     |
| twokeyspressed        | The total time that at least two keys are pressed for.                      | 0.425     |
| shoot-stdtime         | The standard deviation of the times between Shoot events.                   | 0.425     |
| weaponselect-std      | The standard deviation of number of WeaponSelect events per second.         | 0.431     |
| progress-mean         | The average number of Progress events per second.                           | 0.417     |
| weaponselect-bursts   | The number of WeaponSelect bursts per second.                               | 0.419     |
| time-right            | The time the player held +moveright for.                                    | 0.420     |
| all-bursts            | The total number of event bursts.                                           | 0.416     |
| keydown-bursts        | The number of KeyDown bursts.                                               | 0.406     |
| all-num               | The total number of events.                                                 | 0.401     |
| keyup-stdtime         | The standard deviation of time between KeyUp events.                        | 0.405     |
| all-maxtime           | The maximum time between any two events.                                     | 0.404     |
| all-stdtime           | The standard deviation of time between events.                              | 0.404     |
| mean-keyspressed      | The average number of keys pressed at any one time for each key event.      | 0.403     |
| weaponselect-lz-num   | The LZW complexity of WeaponSelect events.                                  | 0.397     |
| Name                  | Description                                                                 | Spearman's $|\rho|$ |
|----------------------|-----------------------------------------------------------------------------|-----------|
| clickburstmovement   | The total distance the player moves while attacking.                        | 0.398     |
| damagespeed          | The time between the player starts attacking and dealing damage.            | 0.393     |
| all-lz-num           | The LZW complexity of any event.                                            | 0.387     |
| shoot-mean           | The average number of Shoot events sampled per second.                      | 0.388     |
| weaponpickup-num     | The total number of WeaponPickup events.                                    | 0.373     |
| shoot-lz-bin         | The LZW complexity of Shoot events as a binary string.                     | 0.371     |
| lz-left              | The LZW complexity of the player’s left key presses (represented as a binary string). | 0.367     |
| mostkeyspressed      | The most number of keys pressed at any one time.                            | 0.365     |
| shoot-maxtime        | The maximum time between any two Shoot events.                              | 0.354     |
| time-backwards       | The time the player held +back.                                             | 0.356     |
| singlemovementratio  | The time that a single movement key was held for divided by the total time movement keys were held. | 0.346     |
| keydown-lz-num       | The LZW complexity of the KeyDown events.                                   | 0.341     |
| keydown-std          | The standard deviation of number of KeyDown events per second.             | 0.328     |
| lz-backwards         | The LZW complexity of the player’s backwards key presses (as a binary string). | 0.325     |
| keyup-bursts         | The number of KeyUp bursts.                                                 | 0.316     |
| progress-stdtime     | The standard deviation of time between Progress events.                     | 0.303     |
| all-lz-bin           | The LZW complexity of events.                                               | 0.301     |
| huffman-keybin       | The Huffman complexity of the key presses represented as a binary string.   | 0.291     |
| Name                   | Description                                                                                                                                                                                                 | Spearman’s $|\rho|$ |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|
| 7  lz-bintostr-keybin  | The LZW complexity of the keys’ binary string (represented by the ASCII code as a binary string).                                                                                                          | 0.286     |
| 6  progress-num        | The total number of Progress events.                                                                                                                                                                        | 0.277     |
| 5  multikeys/3         | The total time for which at least three keys are pressed.                                                                                                                                                     | 0.271     |
| 4  lz-right            | The LZW complexity of the player’s right key presses (as a binary string).                                                                                                                                   | 0.263     |
| 3  time-forwards       | The total time the player held +forward.                                                                                                                                                                      | 0.253     |
| 2  time-left           | The total time the player held +moveleft.                                                                                                                                                                     | 0.249     |
| 1  mean-positionangles | The average angle between the player’s position in the game, represented as a list of vectors.                                                                                                               | 0.230     |