Honours Project Report

Gesture-Based Interaction for Games on Multi-touch Devices

Pierre Benz
Supervisor: Prof. James Gain

	Category	Min	Max	Chosen
1	Software Engineering/System Analysis	0	15	10
2	Theoretical Analysis	0	25	0
3	Experiment Design and Execution	0	20	10
4	System Development and Implementation	0	15	15
5	Results, Findings and Conclusion	10	20	10
6	Aim Formulation and Background Work	10	15	10
7	Quality of Report Writing and Presentation	10		10
8	Adherence to Project Proposal and Quality of Deliverables	10		10
9	Overall General Project Evaluation	0	10	5
Total Marks				80

Department of Computer Science
University of Cape Town
2010.

Abstract

Multi-touch technology has become increasingly accessible with regards to cost and diversity of products available. The ability to understand the capabilities and appropriateness of this technology with respect to computer games is of paramount importance, as games are used for their intrinsic enjoyment value. Similarly, multi-touch enables the user to interact with the device through the use of gestures, which has been shown to provide a more natural means of interaction compared to other traditional input techniques. As games have traditionally been implemented using direct manipulation interfaces, it is crucial to investigate whether gestures can provide a better means of interaction compared to direct manipulation.

The objective of this report is to investigate the use of gestures in games, as well its effects on game genre and identifying user characteristics required for gestural interaction. A gesture recognition engine has been developed, which uses a state-machine with grouped nodes, as well as strategy game, where the objective is to capture three enemy towers using controllable units. User experiments were conducted, where participants were grouped based on their touch device and gaming experience, and played three versions of the game, where each version had either a direct manipulation, gestural or mixed interface. The experiment results show that gestures are the most appropriate means of interacting with games with respect to flow, and that the use of gestures is dependent on game genre. The test also shows that the use of gestures relied on participant touch and gaming experience and not on external factors.

Acknowledgements

I would like to thank my supervisor, Prof. James Gain, for his guidance and helpful comments during the duration of this project, for his detailed feedback, helpful suggestions and numerous corrections, and for his understanding in times of bereavement. I would also like to thank my project group member, Nina Schiff, who not only provided insightful comments during the duration of the project, but also provided a shoulder to lean on during the busy periods of the project. I would like to thank all my friends in CS Honours, for hours of laughter, support and encouragement. Lastly, I would like to my friends and family, for their love, support and encouragement during this year.

Contents

List of Figures 4

List of Tables 6

Glossary 9

1 Introduction 14
- 1.1 Research Questions . . . 15
- 1.2 Project Breakdown . . . 15
- 1.3 Outline . . . 16

2 Previous Work 17
- 2.1 Introduction . . . 17
- 2.2 Multi-touch . . . 17
  - 2.2.1 Definition . . . 17
  - 2.2.2 Multi-touch technology . . . 17
  - 2.2.3 Multi-touch interactions . . . 18
  - 2.2.4 Indirect versus direct-touch interactions . . . 19
  - 2.2.5 Bimanual interactions . . . 19
  - 2.2.6 Multi-touch limitations . . . 19
- 2.3 Direct manipulation . . . 20
  - 2.3.1 Motivation . . . 20
  - 2.3.2 Definitions . . . 20
  - 2.3.3 Limitations . . . 21
- 2.4 Gestures . . . 21
  - 2.4.1 Definitions . . . 21
  - 2.4.2 Limitations . . . 21
- 2.5 Gestures in games . . . 22
  - 2.5.1 Motivation . . . 22
  - 2.5.2 Limitations . . . 23
- 2.6 Methods of evaluating games . . . 23
  - 2.6.1 Motivation . . . 23
  - 2.6.2 Immersion . . . 23
  - 2.6.3 Presence . . . 24
  - 2.6.4 Flow . . . 25
- 2.7 Summary . . . 26

3 Gesture Engine **28**
3.1 Introduction 28
3.2 Gesture Engine Group breakdown 28
3.3 Motivation 29
3.4 Feature Extraction 29
3.5 Linear Node 30
3.5.1 Design 30
3.5.2 Implementation 31
3.5.3 Limitations 32
3.6 State Machine 33
3.6.1 Design 33
3.6.2 Implementation 33
3.6.3 Limitations 34
3.7 Grouped Node State Machine 35
3.7.1 Design 35
3.7.2 Implementation 36
3.7.3 Limitations 37
3.8 Validation 37
3.9 Future Improvements 38
3.10 Conclusion 38

4 Game design **39**
4.1 Introduction 39
4.2 Strategy game 39
4.3 Game design 39
4.3.1 Game-play and key features 39
4.3.2 Units 44
4.3.3 Gesture design 46
4.4 User interface 47
4.4.1 Time-limitation interface 47
4.4.2 Unit interface 47
4.4.3 Minimap interface 48
4.4.4 Direct manipulation, mixed and gesture-based interface 48
4.5 Art and design 50
4.5.1 World design 50
4.5.2 Unit Design 50
4.6 Implementation 51
4.6.1 Technical Specifications 51
4.6.2 Class Layout 52
4.6.3 EAGLView 52
4.6.4 EventHandler 55
4.6.5 GestureEngine 56
4.6.6 Unit 56
4.6.7 Projectile 58
4.6.8 Tower 59
4.6.9 UIManager 59

4.6.10 Renderer 59
4.7 Validation 60
4.8 Conclusion 60

5 Testing and experiments **61**
5.1 Introduction 61
5.2 Motivation 61
5.3 Independent and dependant variables 61
5.4 Participants 62
5.5 Experimental design 62
5.6 Control procedures 62
5.7 Experiment procedure 63
5.8 Pilot experiment 64
5.9 Conclusion 64

6 Results and Analysis **66**
6.1 Introduction 66
6.2 Results of the strategy game 66
6.3 Results of all three games 67
6.3.1 Interface univariate repeated measures results 67
6.3.2 Interface and Genre 70
6.3.3 Interface and Group 70
6.4 Discussion 71
6.5 Summary 73

7 Conclusion **74**
7.1 Future Work 75

8 Appendix A - Experiment forms **76**
8.1 Consent form 77
8.2 Demographic information form 78
8.3 Post game explanation form 79

9 Appendix B - In-game descriptions **80**
9.1 Welcome screen 80
9.2 Training session 80
9.3 Game description 80
9.4 Direct manipulation introduction 81
9.5 After direct manipulation game 81
9.6 Gesture introduction 81
9.7 After gesture game 81
9.8 Mixed introduction 81
9.9 After Mixed game 82
9.10 Experiment end 82

10 Appendix Control - Experiment results **83**
10.1 Strategy game results 83

10.2 Univariate repeated measures 85
10.3 Interface and game genre 89
10.4 Interface and participant group 91

References **94**

List of Figures

1.1 An overview of the project breakdown between group members 16

2.1 An example of two multi-touch interaction styles. (a) shows the use of bimanual interaction, while (b) shows the use of multi-finger interaction. 18
2.2 An example of a direct manipulation interface, where the user interacts with the interface through buttons. 21
2.3 An example of a gestural interaction. 22

3.1 An overview of what each group participant contributed to the gesture engine. 28
3.2 An example of two different feature extraction methods used in the gesture engine. Both feature extractions are performed on ’V’ and ’U’ gestures. a) shows the use of parabolic curve-fitting feature extraction, while b) shows the use of the change-in-direction feature extraction. Note that the parabolic curve-fitting does not recognise the initial vertical line drawn, which often resulted in ’U’ gestures having the same features as ’V’ gestures. 30
3.3 An example of the *Linear Node* gesture engine, where each parent node has only one child . . 31
3.4 The direction map the *State Machine* gesture engine uses to map features. Also shown is the zones in which features are rounded. 33
3.5 An example of a spiral gesture being performed with feature extraction displayed as coloured lines. a) multiple ways of extracting features exist, which shows the trouble in easily distinguishing a spiral gesture. B) displays a solution the *Grouped Node State Machine* uses to address the curved features present in the spiral gesture. 35
3.6 An example of the gesture nodes present in both the *State Machine* and *Grouped Node State Machine* for a spiral gesture. As can be seen, a) has multiple gesture definitions to a handle a single gesture, whereas b) shows a single gesture definition which handles all possible variations. 36
3.7 An example of two *StateMachineNodes* which are used in the *Grouped Node State Machine*, where a check is made against the current node’s directional values, after which a check is made against the child node. 37

4.1 An example of the direct manipulation interface showing the placement area after a player selected a foot-soldier to be built 41
4.2 Interaction circles of two units, where (a) shows a foot-soldier within the interaction-circle of a archer, causing the the archer to attack the foot-soldier, and (b) shows a foot-soldier and archer within each others interaction-circles, where both units will attack each other 42
4.3 An example of an attack sequence. A red line is drawn between the archer and the foot-soldier, with a pulsing animation shown in the consecutive images. In step d) the archer has waited sufficient amount of time to perform an attack check, where in steps a, b and c the red circle shows how much the archer still has to wait in order to perform an attack. 43

4.4 List of gesture available in the gesture-based and mixed game interface 46

4.5 Time-limitation interface, where a coloured bar indicates the remaining time the player has to complete the game and the severity of the total time left 47

4.6 A display of the different interfaces the game will display depending on the game style. a) shows the direct-manipulation interface, where buttons allow the player to interact with unit based commands, while b) show a mixed interface, where some direct-manipulation buttons are still availabe and c) shows a gesture-based interface, where all button have been removed as the interaction are all gesture based. 48

4.7 Minimap interface which is located at the bottom left of the interface 49

4.8 An example of a direct manipulation interface, showing a foot-soldier being selected by the player. Notice that the map movement button are displayed on the left, right, top and bottom of the game map and will move the screen in either direction depending on which button has been pressed. 49

4.9 An example of a mixed interface, showing a foot-soldier being selected by the player. A list of context-sensitive gestures are displayed on the right-hand side of the screen. 50

4.10 An example of a gesture-based interface, where a group one selection gesture has been performed. A list of context-sensitive gestures are displayed on the right-hand side, as well as the previously performed gesture that is show in the top right corner 51

4.11 An overview of the main game environment, displaying the playing environment where game-play will occur and where the towers will be located. The blue rectangle displays the size of the display screen. 51

4.12 An example of how player and enemy units are displayed in the game. Player controllable units have a blue circle surrounding the unit, while enemy units are surrounded by a red circle. . . . 52

4.13 Representation of the class hierarchy used in the game. The lines between the nodes represent how each class used other classes, where the arrows indicates the direction of class use. 52

4.14 The game total game environment broken up into nine 500x500. Note that the game screen, indicated by the blue rectangle, overlaps over the nine regions. 54

4.15 Internal representation of the *GestureStateMachine* object nodes that were used in the game. . 56

List of Tables

3.1 Results from gesture engine comparison. Percentages of classification in different categories. . . 38

3.2 Statistical T-test results performed on the gesture engine comparisons. 38

4.1 The cost of each unit available in the game 40

4.2 Unit Interaction-circle radii 42

4.3 A summary of the projectiles associated with units and their specifications. 44

4.4 A summary of the units available and their specifications. 44

4.5 List of units available in the game 45

4.6 Unit style and design 50

6.1 A summary of the different dimensions of flow 67

6.2 Multivariate tests of significance on all three game. Significances are shown in red. 68

6.3 A summary of the results found for all three games. 69

10.1 Total mean flow score between different game interfaces 83

10.2 Breakdown of the flow dimensions between different game interfaces 83

10.3 Univariate tests for repeated measures strategy game 84

10.4 Repeated measures analysis of variance on the strategy game 84

10.5 Univariate for repeated measure on *challenge-skill balance* flow dimension using interface, group and gesture independant variables 85

10.6 Tukey HSD test for *challenge-skill balance* flow dimension. Error: within MS = 6.9370, df=72.000. Significant values are shown in red. 85

10.7 Univariate for repeated measure on *action-awareness merging* flow dimension using interface, group and gesture independant variables 85

10.8 Tukey HSD test for *action-awareness merging* flow dimension. Error: within MS = 6.1981, df=72.000. Significant values are shown in red. 86

10.9 Univariate for repeated measure on *clear goals* flow dimension using interface, group and gesture independant variables 86

10.10 Tukey HSD test for *clear goals* flow dimension. Error: within MS = 6.2667, df=72.000. Significant values are shown in red. 86

10.11 Univariate for repeated measure on *unambiguous feedback* flow dimension using interface, group and gesture independant variables 86

10.12 Tukey HSD test for *unambiguous feedback* flow dimension. Error: between; within; Pooled MS = 9.6370, df=95.455. Significant values are shown in red. 86

10.13 Univariate for repeated measure on *concentration on task at hand* flow dimension using interface, group and gesture independant variables 87

10.14 Tukey HSD test for *concentration on task at hand* flow dimension. Error: within MS = 10.756, df=72.000. Significant values are shown in red. 87
10.15 Univariate for repeated measure on *sense of control* flow dimension using interface, group and gesture independant variables 87
10.16 Tukey HSD test for *sense of control* flow dimension. Error: within MS = 6.8074, df=72.000. Significant values are shown in red. 87
10.17 Univariate for repeated measure on *loss of self-consciousness* flow dimension using interface, group and gesture independant variables 87
10.18 Tukey HSD test for *loss of self-consciousness* flow dimension. Error: between; within; Pooled MS = 9.6370, df=95.455. Significant values are shown in red. 88
10.19 Univariate for repeated measure on *transformation of time* flow dimension using interface, group and gesture independant variables 88
10.20 Tukey HSD test for *transformation of time* flow dimension. Error: within MS = 9.8037, df=72.000. Significant values are shown in red. 88
10.21 Univariate for repeated measure on *autotelic experience* flow dimension using interface, group and gesture independant variables 88
10.22 Tukey HSD test for *autotelic experience* flow dimension. Error: within MS = 10.506, df=72.000. Significant values are shown in red. 88
10.23 Tukey HSD test for *challenge-skill balance* flow dimension. Error: between; within; Pooled MS = 10.256, df=89.300. Significant values are shown in red. 89
10.24 Tukey HSD test for *action-awareness merging* flow dimension. Error: between; within; Pooled MS = 10.115, df=83.085. Significant values are shown in red. 89
10.25 Tukey HSD test for *clear goals* flow dimension. Error: between; within; Pooled MS = 10.667, df=80.578. Significant values are shown in red. 89
10.26 Tukey HSD test for *concentration on task at hand* flow dimension. Error: between; within; Pooled MS = 13.300, df=100.63. Significant values are shown in red. 90
10.27 Tukey HSD test for *sense of control* flow dimension. Error: between; within; Pooled MS = 11.300, df=82.059. Significant values are shown in red. 90
10.28 Tukey HSD test for *transformation of time* flow dimension. Error: between; within; Pooled MS = 14.722, df = 88.291. Significant values are shown in red. 90
10.29 Tukey HSD test for *autotelic experience* flow dimension. Error: between; within; Pooled MS = 18.648, df = 78.186. Significant values are shown in red. 90
10.30 Tukey HSD test for *challenge-skill balanca* flow dimension. Error: between; within; Pooled MS = 10.256, df = 89.300. Significant values are shown in red. 91
10.31 Tukey HSD test for *action-awareness merging* flow dimension. Error: between; within; Pooled MS = 10.115, df = 83.085. Significant values are shown in red. 91
10.32 Tukey HSD test for *clear goals* flow dimension. Error: between; within; Pooled MS = 10.667, df = 80.578. Significant values are shown in red. 91
10.33 Tukey HSD test for *unambiguous feedback* flow dimension. Error: between; within; Pooled MS = 9.6370, df = 95.455. Significant values are shown in red. 92
10.34 Tukey HSD test for *concentration on task at hand* flow dimension. Error: between; within; Pooled MS = 13.00, df = 100.63. Significant values are shown in red. 92
10.35 Tukey HSD test for *sense of control* flow dimension. Error: between; within; Pooled MS = 11.300, df = 82.059. Significant values are shown in red. 92
10.36 Tukey HSD test for *loss of self-consciousness* flow dimension. Error: between; within; Pooled MS = 17.422, df = 82.800. Significant values are shown in red. 92

10.37 Tukey HSD test for *transformation of time* flow dimension. Error: between; within; Pooled MS = 14.722, df = 88.291. Significant values are shown in red. 93
10.38 Tukey HSD test for *autotelic experience* flow dimension. Error: between; within; Pooled MS = 18.648, df = 78.186. Significant values are shown in red. 93

Glossary

Analysis of variance [ANOVA] A statistical test used to identify any difference between groups of variables. It provides a statistical test for whether the means of several groups are equal. As such, it generalises t-tests to two or more groups. A repeated measures ANOVA is also used to address the issue of the same subjects being used in each treatment. *See* Chapter 5.

Application Programming Interface [API] An interface used in computer programming languages which enables it to be used with other software libraries and applications.

Bimanual interaction Interactions done with the use of both hands. *See* Chapter 2.

Dependent variables A variable which is dependent on another variable. This type of variable is often used in mathematics and statistics. When used in conjunction with an independent variable, it allows the ability to distinguish between two types of quantities and what they produce. It should be noted that the independent variables are the variables that are manipulated, while the dependent variables are only measured. *See* Chapter 4 and 5.

Direct manipulation An interaction style which uses a representation of objects and actions, where simplified and reversible actions replace overly complex ones that are continually visible on a interface (commonly implemented on a user interface as a button widget). *See* Chapter 2.

Direct touch An interaction with the ability to touch a single specific location on the display directly with the use of a finger or a stylus. *See* Chapter 2.

Duelling games A video game genre that emphasises action and fighting challenges. The game-play usually involves one or more playable character, with the aim to eliminate an opposing unit their a means of combat.

Electromyography [EMG] The technique of evaluating and recording the electrical activity produced by skeletal muscles of a applicant. *See* Chapter 5.

Factorial design An experiment whose design consists of two or more factors, where each factor consists of a discrete set of values. The aim of a factorial design is to address all possible combinations of these values across all factors in an experiment. *See* Chapter 4.

Feature extraction Method used to simplify and describe a pattern of resources necessary to describe a large set of data. Feature extractions is typically used in image processing. *See* Chapter 3.

Flow The state where a successful balance between a user's ability and the perceived challenge or difficulty exists and is associated with the experience that is in itself so gratifying that the user will be willing to do it for its own sake, with little concern for what they might be getting out of it in return or if it was difficult. *See* Chapter 2.

Flow State Scale [FSS] A questionnaire used to measure flow experienced by a participant and is commonly used in sport and physical activity settings. *See* Chapter 5.

Gestures An interaction style which uses a symbolic method of giving a command to a computer, where its location, size and dynamic properties play a role in its differentiation, and is defined as being a more natural means of interacting with a computing device. Gestures are usually drawn using an input device. *See* Chapter 2.

Gesture recognition A system used to interpret and recognise gestures performed by a person. *See* Chapter 3.

Hidden markov model [HMM] A statistical Markov model, where a system is composed with unobservable states. Only the output of each state is visible. Each state is composed of a probability distribution over the possible output tokens. As such, the sequence of tokens generated by the Hidden Markov Model gives information of the series of states. HMMs are typically used in pattern recognition systems. *See* Chapter 3.

Immersion The psychological state in which the immersant's awareness of their physical self is diminished or lost due to the level of stimuli produced from the use of a technological system. *See* Chapter 2.

Independent variable A variable which is independent of other variables. This type of variable is often used in mathematics and statistics, when used in conjunction with an dependent variable, it allows the ability to distinguish between two types of quantities and what they produce. *See* Chapter 4 and 5.

Independent Television Commision's Sense of Presence Inventory [ITC-SOPI] A questionnaire used to quantify presence through the use of four independent sub-scales, namely sense of physical space, engagement, ecological validity and negative effects. *See* Chapter 2.

Indirect touch An interaction with the use of a pointing device, such as a mouse or a track-pad, where the user interacts with the device off-screen and has to visually track a cursor on-screen. *See* Chapter 2.

Integrated development environment [IDE] An application or environment that provides a number of facilities to computer programmers for software development. An IDE usually consists of a source code editor, compiler and debugger. *See* Chap

Likert scale A psychometric scale used in questionnaires, where the respondent specifies their level of agreement with a statement. For example, a 5-point scale will have the following levels: strongly disagree, disagree, neither agree nor disagree, agree, strongly agree. *See* Chapter 5.

López-Dahab [LD] coordinates Coordinates used to represent elliptical curve points on binary curves $y^2 + xy = x^3 + ax^2 + b$. The triple (X, Y, Z) represents the affine point (X / Z, Y / Z^2).

Multivariate analysis of variance [MANOVA] A statistical test used when more than one dependant variable is present. As well as identifying whether changes in the independent variables have significant effects on dependent variables, they are also used to identify the interactions between dependent and independent variables. *See* Chapter 5.

Multi-touch The use of two or more fingers as input onto a touch screen. *See* Chapter 2.

Multi-finger interaction The use of two or more fingers on a computing device. *See* Chapter 2.

Open Graphics Library [OpenGL] A cross-platform, cross-language graphical library used to produce 2D and 3D computer graphics. *See* Chapter 3.

Presence The illusion where the participant believes the environment that they are being presented with, through some medium, is real. *See* Chapter 2.

Puzzle games A video game genre that emphasises strategic, logical, pattern recognition and sequence solving challenges. The game-play usually involves shapes, colours or symbols which must be directly or indirectly manipulated to form a specific pattern or objective.

PVRTC A lossy image format optimised for use in PowerVR MBX technologies. *See* Chapter 3.

Significant value A results that is unlikely to have occurred by chance and is used extensively in Statistics. *See* Chapter 5.

State-machine A system with a finite number of states, which is used to transition between states in a system. The state machine moves from state to state when certain conditions are met, typically defined by each state and the system running the state machine. The states are composed of arbitrary data structures and are set together with a number of functions. State-machines are typically used in parsing systems. *See* Chapter 3.

Strategy computer game A video game genre that emphasises strategic, tactical and logical challenges. The goals of the game vary from eliminating all opposing units to capturing targets. Most strategy games involves a certain amount of warfare, as well as the use of environment exploration and resource management. *See* Chapter 3.

Type 1 error (false positive) The error of rejecting a null hypothesis when it is actually true. *See* Chapter 3.

Type 2 error (false negative) The error of failing to rejecting a null hypothesis when a null hypothesis did occur. *See* Chapter 3.

Unimanual interaction Interactions done with the one hand. *See* Chapter 2.

Usability tests The evaluations of a product by testing it on users. The aim of the tests is to systematically observe how people interact with a product to discover errors or areas of improvement. Tests are measured in terms of user performance, accuracy, recall and emotional response.

Virtual reality The use of computer-simulated environments to simulate places, both real and imaginary. These environments are primarily used as visual experiences and are displayed through computer screens, projectors or stereoscopic displays.

Chapter 1

Introduction

We use our hands to interact with objects; We pick them up, move them, transform their shape and interact with them [44]. In short, we use out hands to interact with the world [17]. Computing devices have become a foundational element in our society and have made numerous advantages to our day-to-day lives. People spend most of their time inputting information into computers, yet the means with which we interact with these devices have not changed in years. As the most useful information does not reside in computing devices but in people [27], the need to optimise the way we interact with them is of importance. If not, the time spent on physically performing tasks on computing devices will not improve, even if the time it took to process that information was to become infinitely fast [17]. While two modes of interaction have become the most prominent in most interactive computing devices, namely gestures and direct manipulation, they differ in their means of display and interaction. Gestures refers to a symbolic sequence of actions that are drawn onto the device using an input medium, such as a finger, and provides a higher level of abstraction between the interface and commands issued from the interface. Direct manipulation refers to the usage of constantly visible objects and actions, generally in some form of a button, which is used to replace overly complex syntax and commands.

With the rise in availability of computing devices in everyday society, the computer game industry has also been seeing significant growth over the last few decades. While this growth has equalled and at times exceeded the film industry with respect to revenue, little is known about the factors that contribute to a successful game [26]. While standard usability tests could be performed on games, it overlooks a crucial element that separates games from conventional software, which is that games are used as entertainment and not as utility [42]. As such, usability tests need to be modified to include research from other fields, such as virtual reality, in order to properly evaluate computer games.

Multi-touch technologies allow the user to interact directly with the elements presented on the screen, without the use of external input technologies. It also supports the use of two or more fingers as input, allowing direct interaction with the device. The benefits of this is that it elicits the feeling of being more “natural” and “compelling” compared to indirect methods of interaction [14]. This affordance could result in improving efficiency and accuracy in interaction. One of the advantages of multi-touch interactions is that they are intuitive and easy to learn. The interface designer can meet the user’s expectations by maintaining an analogy to the physical world [31]. This also opens up many possibilities for novel solutions that were previously constrained with single pointer technologies. Multi-touch technology also enables the ability for two or more people to interact on the same device. This allows for collaborative interactions, which could not only change the way we interact with devices but also with the way we interact with others.

1.1 Research Questions

This research project aims to address the question of whether gesture-based interaction provides a better means of interaction in games compared to direct manipulation. This is important as there is a need to identify whether gestures improve interaction when compared to more conventional techniques such as direct manipulation. In order to answer this question, three version of a game will be created, namely a direct manipulation, gestural and mixed interface (the mixed interface will contain a combination of direct manipulation and gestural interaction methods). Similarly, two smaller questions are investigated in order to provide an adequate analysis of the main research question. These questions are as follows:

Gesture effects on game genres An investigation as to whether gestures are an appropriate input mechanism for different game genres is needed. In addition, an examination of how the complexity of gestures affects the game-play experience within each genre and whether there are genre-dependant differences with the usage of gestures. In other words, if there are genre differences, how does this affect the appropriateness of gesture specifics? This is important to distinguish, as some games may lend themselves to gestures more than other genres.

Gesture effects of participant characteristics By investigating which participant characteristics influence the usage of gestures within games, an accurate analysis as to what the contributing factors for successful gestural use in games is. As such, the investigation aims to investigate whether touch device and gaming experience play a role in positive gesture game interactions. This is important, not only in analysing why gestures are an appropriate means of interaction, but also providing insights for future research in gestural interactions in games.

1.2 Project Breakdown

The project is broken up into three sections, namely the gesture engine, games and experimental design. Figure 1.1 shows the project breakdown, with project member sections displayed in a different colour. The gesture engine contains a feature extraction subsystem [39] and two gesture recognition engines, namely a Hidden Markov Model [49] and a state-machine gesture engine. That latter is outlined in this report. It allows the gesture, which are performed by the user, to be recognised by the device.

There are a total of three games developed, each game focusing on a different game genre. The game genres include a puzzle game [39], a duelling game [49] and a strategy game, which is described in this report. The games developed here are the games that are going to be used in the experiments.

Lastly, the experiments that are going to be performed are developed by Nina Schiff [39] and includes the experimental process as well as the experimental techniques.

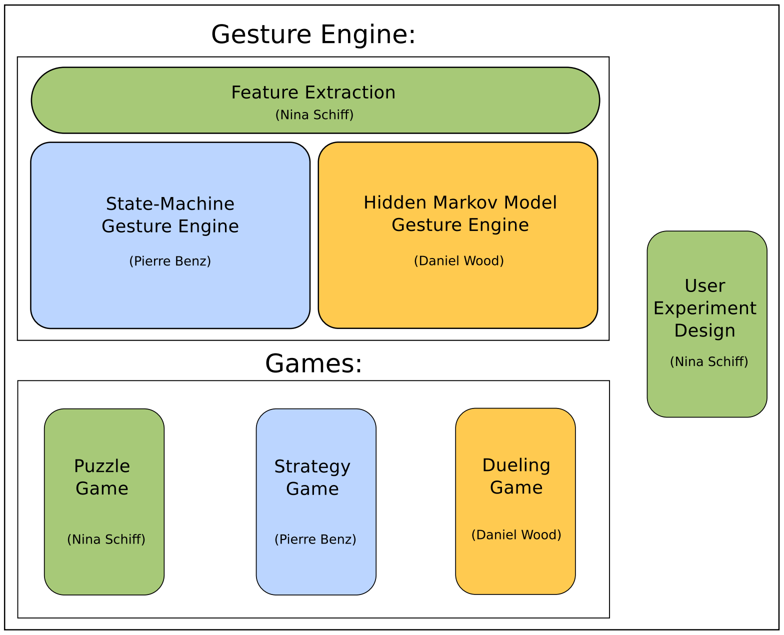


Figure 1.1: An overview of the project breakdown between group members

1.3 Outline

In Chapter 2 the previous work pertaining to this project is discussed. This is followed by the design, implementation and validation of the state-machine gesture engine in Chapter 3. The game design, implementation and validation is present in Chapter 4. Chapter 5 presents the design and procedure of the user experiments carried out, while Chapter 6 presents and discussed the results of the experiment. Finally, Chapter 7 concludes with the main results of this project and possible future work.

Chapter 2

Previous Work

2.1 Introduction

This chapter will take a brief look at the relevant previous work done that pertains to this project. This includes, among other things, a brief synopsis of multi-touch technology and interaction styles, direct manipulation, gestures, gestures in games and what evaluation methods are suitable for games.

2.2 Multi-touch

2.2.1 Definition

Multi-touch is defined as the simultaneous use of two or more fingers as input onto a touch screen [9]. This allows for the removal of external input means, such as a mouse or a track-pad, leaving a direct interaction with the screen. The nature of the interaction is direct, meaning that the user either interacts with the interface or does not. The benefits of multi-touch is the notion that interacting with an application through touching graphical elements with one's hands provides a more "natural" and "compelling" means of interaction compared to indirect input mechanisms [14]. This might also results in improved efficiency and accuracy with regards to interacting with devices [31]. Since objects move and are interacted in a predictable and natural fashion, users are given the impression of "gripping" real objects [36].

2.2.2 Multi-touch technology

Currently, there is no standard hardware on which multi-touch interfaces are developed. Some devices are based on camera systems, while other employ capacitive technologies. This has lead to multiple design approaches based on the capabilities of the devices [45]. For example, Apple's track-pad allows multi-touch gestures, including a four finger touch gesture, while the HP TouchSmart can only track two points of contact. It should also be noted that not all touch-based technologies support multi-touch. In this case, they only allow single actions to be performed sequentially. Different devices also have support for different input mechanisms; some allow the user to interact directly with their fingers, such as the Apple iPhone, while others support the use of a stylus, such as Microsoft Window 6.5.

Multi-touch interactions are very well suited for object manipulation tasks [29]. By using

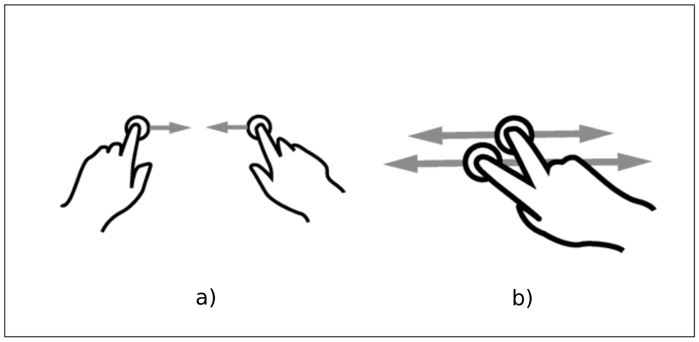


Figure 2.1: An example of two multi-touch interaction styles. (a) shows the use of bimanual interaction, while (b) shows the use of multi-finger interaction.

the analogy of physical manipulation as a model for mapping from hand positions to object parameters, one can construct easy-to-learn techniques that leverage our real-world interaction experience. One of advantages of touch-screen interactions is that they are intuitive and easy to learn. The interface designer can meet the user's expectations by maintaining an analogy to the physical world [31]. This opens up many possibilities for novel solutions that were previously constrained with single pointer technologies.

The user interacts with the device by touching the screen and selecting objects with their fingers. This type of interaction leaves the assignment of a finger to specific regions up to the person using the device. Although this interaction frees the user to interact with the device as they chose, it has implication for designers of multi-touch interaction techniques and interface designers. To date, no sufficient design patterns have emerged which adequately describe gestural use patterns [31].

2.2.3 Multi-touch interactions

Interactions on a multi-touch device can be broken up into three groups, namely direct-touch, bimanual and multi-finger interactions [23]. Direct-touch interaction refers to the ability to touch a single specific location on the display directly with the use of a finger or a stylus. In contrast, an indirect interaction refers to the use of a pointing device, such as a mouse or a track-pad, where the user would interact with the device off-screen and would have to visually track a cursor on-screen. Bimanual interaction refers to interactions done with the use of both hands, while multi-finger interaction refers to the use of multiple fingers but not necessarily the use of multiple hands.

The advantages that multi-touch brings to interaction are numerous. It allows for parallel interactions, meaning that interactions can be performed at the same time, which reduces the complexity required in user interfaces [29]. Another benefit of parallelism is that it potentially leads to faster performance, as the execution of sub-tasks could overlap instead of performing them sequentially. Moreover, each hand can remain in the area of proximity it is currently operating, reducing the movement required [11]. Multiple fingers also allow the user to grasp or reach several objects at once. For example, multiple fingers might be used to control an array of sliders in a user interface. Similarly, multiple hands allow user to perform better at tasks requiring separate control points than using only one hand. This has also been shown to be the case when control points are within the rage of one hand's fingers [31]. As described in section 2.2.5, there are limitations in the degree of independent freedom of fingers with respect to multi-touch interactions, but these are discussed there.

2.2.4 Indirect versus direct-touch interactions

In a single user setting, direct-touch interactions offer several benefits over indirect interactions. Where indirect interactions require the user to visually track an onscreen cursor in a separate position from the hand, a direct-touch device offers interaction where the input and display are co-located[23].

In an investigation done by Sears and Schneiderman [40] to test whether interaction target size effects speed and accuracy between indirect and direct-touch interaction techniques, it was found that direct-touch interaction speeds were faster for targets of size 0.64cm in width or larger than with a indirect interaction. Users interacting with target sizes of 1.28cm made 66% fewer error with the touch screen than with a mouse. It was also shown that direct-touch interaction for single target selection tasks outperformed indirect mouse-based interactions, however, occlusion due to the imprecision of using fingers as an input mechanism to locate touch-points was a major factor in the accuracy reduction of direct-touch interaction [23]. A similar investigation done by Forlines et al [14], which focused on unimanual interaction, showed that single-target selections tasks with target sizes of 1.92cm and larger were significantly faster for direct-touch interactions compared to a indirect, mouse-based interaction. However, the direct-touch interaction also produced twice the amount of errors compared to the indirect interaction.

2.2.5 Bimanual interactions

Users are better at selecting and dragging objects using a mouse or track-pad than using one or two hands on a multi-touch device, however, these tasks generally require pixel-precision in their selection and manipulation, which is very difficult to achieve with direct-touch interaction [40]. Bimanual interactions were also shown to be faster than using unimanual interactions, but they tend to have higher error rates while performing tasks, such as target selection [23]. Even though these errors occurred more frequently in bimanual interactions, the time it took to correct mistakes made and complete the tasks is faster compared to unimanual interactions. Unfortunately, only selection tasks have been sufficiently researched, leaving other tasks commonplace to user interface interactions needing research [31].

Performance comparisons between the use of two fingers on one hand compared to the use of one finger on each hand shows that one fingered, two-handed interactions are better for performing separable tasks, such as manipulating separate control points on a graphical object, whereas two fingered, one handed interactions are better at integral tasks, such as rotations of two-dimensional objects [31].

2.2.6 Multi-touch limitations

While multi-touch devices can detect multiple points of input, allowing the design of high-degree-of-freedom interaction techniques that make use of our real-world manipulation abilities, they are not without their limitations. Because of the physical limitations of the human hand, multi-touch devices suffer from the limited precision, occlusion and limitations on the size and proximity of the actual multi-touch display [31]. The hand and fingers often obscure the very object the user is interacting with. When a finger is used to select an object, the hand may cover certain areas of the screen, making it very difficult to interact with other areas of the screen, as well as hiding what may be crucial aspect of the interface. In contrast, if a user uses an indirect interaction, such as a mouse, no part of the screen would be obstructed. In many cases, fingers are wider than many user interface widgets common today, making precise selection difficult [31]. At the same time, it must be taken into account that it is difficult to accurately point at objects that are smaller than one's finger [14]. Simply making targets bigger, so that interactions are easier, may not be possible as it reduces the amount of content displayable on the screen.

A reasonable way of increasing the interaction on multi-touch surfaces is with the use of two hands. Two-handed interaction is especially well suited for high-degree-of-freedom symmetric bimanual tasks, such as shape deformation. However when two interactions are too close together it becomes difficult to distinguish which point on the touch devices belongs to which finger [30]. Fingers on the same hand inherit the hand's motion, meaning their movement may be perceived as being relative to the hand's frame of reference. The motion of the fingers on separate hands can be more easily uncoupled than that of fingers on a single hand. However, while the motion of fingers on a single hand are easier to coordinate than that of fingers on opposing hands [31].

Error rates, corresponding to target selection tests, have been shown to have similar results across all four input methods. However, bimanual interactions tend to have a higher error rate than unimanual interactions. There was also a difference of 6.5% in error rates between one finger and multiple finger interactions. The high error rates for multi-finger interactions may be due to the increase in cognitive load required to plan movements for multiple fingers independently. In addition, fingers on the same hand are physically constrained to the palm, which limits the set of targets fingers can access simultaneously [23].

Finally, there is also a reason why early scribes began writing with a reed rather than dipping their finger in ink. The same applies to multi-touch technologies. This is because it is easier to control and produce lines of higher resolution when using a pen or stylus[30]. It is important for designers to take the context, use and practicality of each input mechanism into account [45]. This means that in order to take full advantage of multi-touch many applications would have to be redesigned, as applications mainly only handle sequential interface actions.

2.3 Direct manipulation

2.3.1 Motivation

2.3.2 Definitions

Direct manipulation is defined as the representation of objects and actions, where actions replace overly complex syntax and reversible operations are immediately visible on the interface [41]. The benefits of direct manipulation are manifold. Users unfamiliar with the interface can learn the basic functionality very quickly, while experts can perform a wide range of tasks through successive interactions and can retain operational concepts. Error messages are rarely needed as users can immediately see if their actions are successful, meaning users are less anxious when interacting with the interface since actions are comprehensible and easily reversible. Finally, users become confident using the interface, as they feel in control [41]. Examples of direct manipulation in modern user interfaces are a graphical button or draggable objects, which the user interacts with directly.

Figure 2.2: An example of a direct manipulation interface, where the user interacts with the interface through buttons.

2.3.3 Limitations

One of the limitations that direct manipulation imposes is that it rises or falls with the quality of the selected descriptive or graphical representation of the underlying action. This means that the action being represented by the direct manipulation interface must be meaningful and accurately signify the action it controls [15]. This directly effects the learnability and memorability that direct manipulation affords, as it will also depend on the users prior knowledge of the action at hand and its semantic mapping to the system.

One of the benefits that direct manipulation has over long, complex syntactical commands is that it makes complex commands more usable to novice users. Paradoxically, this has a negative effect on expert users, as the loss of language, which is inherent in direct manipulation interfaces, also means that there is a loss of descriptive power for different classes of actions. Examples of this can be found in user interfaces, where parsing the interface for advanced commands slows down the time to efficiently execute commands. Similarly, the use of familiar real-world metaphors might be beneficial to novice users but not for expert users, who might be familiar with different metaphors than those being presented [15].

2.4 Gestures

2.4.1 Definitions

A gesture is roughly defined as a symbol used to give a command to a computer, where the attributes of a gesture, such as its location, size, extent, orientation and dynamic properties are mapped to parameters of that command [37]. Because of this, gestures allow for a higher-level of control over commands, allowing interfaces to be more streamlined, as there is no need for overly-complex user interface structures. As McNeill [32] describes, “Indeed, the most important thing about gestures is that they are *not* fixed. They are free and reveal the idiosyncratic imagery of thought”.

2.4.2 Limitations

One of the biggest limitations of gestural interactions is that they are usually completed before the gesture can be classified. This means that the attributes that define and differentiate a gesture from others are only examined once the gesture has been completed, leaving no oppor-

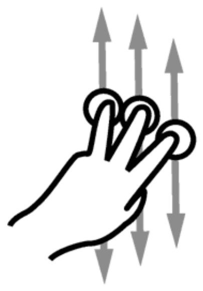


Figure 2.3: An example of a gestural interaction.

tunity for the gesture to be cancelled or to be manipulated into a different gesture. This has been described as a lack in continuous feedback and has contributed to the awkwardness that gestures have in performing common tasks [37]. Similarly, the lack of feedback affects the user in a negative manner, as the user might not be sure how they have commanded the system [48].

Another limitation that surrounds gestures is not necessarily due to properties of gestures, but in the systems that classify and differentiate them. To date, gestures have primarily been defined by system designers and technicians, employing them and teaching them to users. Most of the time, these gestures are arbitrary and are only chosen due to reliable recognition that the underlying engine can perform [48]. Unfortunately, no current system exists which can recognise fully dynamic gestures [44], which does affect the design of gestures. Recognition versus recall also plays a role in gestures, as one could design a system in which all commands were executed with the use of a gestural interaction, but would be very difficult to learn [50].

Finally, while gestures might provide a more natural interaction compared to that of direct-manipulation, often the simplest of gestures are not simple enough. An example of this is a swipe gesture, where a finger moves from one side of the screen to the other side to perform a ”move back” command, which is more complicated that pressing a button. There are many cases where direct manipulation would be more intuitive and more time saving than performing a gesture.

2.5 Gestures in games

2.5.1 Motivation

Several past explorations in the usage of gestures in games has been performed, and would be invaluable in this project, as the focus of the project is in the appropriateness of gestures in games and within game genres. The advantage of using gestures in games are numerous. They replace cumbersome user interface structures and intrusive controllers, allowing players to feel in control over their actions [34]. When combined with external, physical apparatuses, they allow physical freedom of motion, which has become a very attractive scenario for new gaming experiences.

Several commercial games have utilised gestures within games, such as Lionhead Studios’ *Black & White*, where the player uses gestures to cast spells by utilising the mouse. The usage of gestures in such games removes the use of menu structures, allowing the player a more flexible and enjoyable game experience, and allows the player to focus on the object being manipulated instead of interface objects.

Most past work being done in the field of gesture-based games gestures are with the use of commercial external apparatuses, such as the Nintendo's Wiimote, Sony's EyeToy or with Microsoft's Kinect. Other mechanisms involve the use of camera systems, where the user is tracked in the cameras viewport. Gesture engines are usually implemented by using a statistical model, such as a Hidden Markov Model [24], or with a gesture spotting method, which is frequent in camera-based systems, where gestures are distinguished from unintentional movements [22].

2.5.2 Limitations

While gestures do provide a new and exciting mechanism for the game-play experience, it has not been proven to provide a new game-play paradigm nor is it without it's limitations [46].

While it is a new and exciting paradigm for the traditional method of interacting with games, it is not without its limitations. The use of physical gestural apparatuses, while allowing for a more natural method of interacting with a game, brings new issues, such as user feedback and human spatial motion awareness. While gestures can take the form of real world metaphors, players preferred simple gestures, where immediate context and actions were identifiable. Similarly, teaching of gestures and remembering what each gesture does severely affected game-play, at times frustrating players when critical game-play sections are required. This often results in gesture confusion, and as a result, gesture engines that rely on this method have the trouble of distinguishing similar looking gestures from one another [34]. Camera systems also the have the drawback of being sensitive to the environment the player is playing the game, as the surrounding environment needs to successfully contrast the player who is using the device.

In cases where gestures required the use of statistical gesture engines, such as Hidden Markov Models, the accuracy of the gesture being recognised was largely dependant on the time spent training the gesture engine to the specific player [24]. This severely affects the usage of games that rely on get-up-and-go game-play, where players generally do not want to spend time training a game to be calibrated to their method of gesture input. Difficulty is often found when using the gesture spotting method, as multiple variations of player posture significantly affects which gesture are recognised.

2.6 Methods of evaluating games

2.6.1 Motivation

Games are fundamentally different from general purpose software, as their main purpose is to entertain instead of being a utility to accomplish a specific task [42]. This means that standard usability evaluations will only address a small subset of features in games. As such,psychological findings and evaluation methods are needed to properly asses games.

2.6.2 Immersion

Immersion is defined as the psychological state in which the immersant's awareness of their physical self is diminished or lost due to the level of stimuli produced from the use of a technological system [28]. It has been shown that there are a number of immersion levels through which an immersant proceeds before reaching total immersion. While there are different labels individuals have given for each different stage, such as absorption, engagement and involvement, all these labels describe a state in which the immersant is focused on the experience at hand, whereas total immersion is described as the experience of being wholly engaged in the experience and completely involved in this separate reality[12].

However, total immersion is very difficult to achieve [47]. This is because immersion requires the immersant to be fully engaged and overlooking of the artificial reality they are interacting with. This engagement is often done with the use of an external interface and is only possible in a virtual reality environment, where the goal is to fully immerse the immersant [13].

Measuring immersion

While there has been plenty of discussion of the effects of immersion and what it could mean, there is a lack in quantitative methods for evaluating the different levels a immersant might be experiencing.

A test developed by Witmer and Singer [47] measures the inclination an immersant has to become immersed in an environment. They came up with the Immersive Tendencies Questionnaire[ITQ], which uses three dimensions, namely *involvement*, *focus* and *games*, to test the immersive inclination an immersant might have. Involvement assesses the immersants' inclination to become immersed in a medium, while focus measures the degree an immersant is able to concentrate on one task. Games measures how often a immersant plays games and how involved they became immersed. The test shows reliability and validity, as well as a a high internal consistency rate ($\alpha = 0.81$). While this test does not directly measure the amount of immersion an immersant experiences in a game environment, it does show how readily an immersant experiences presence, with a significant positive correlation ($r = 0.24$, $p < 0.01$) [47].

Similarly, a test developed by Ermi and Mayra [12] identifies three dimensions of immersion, namely *sensory immersion*, *challenge-based immersion* and *imaginative immersion*. Sensory immersion describes the audio-visual experience of the environment, challenge-based immersion describes the combination of the immersant's abilities and the challenge presented in the environment, and imaginative immersion describes the narrative dimension of the environment experience[12].

Finally, there appears to be no standard way of measuring the amount of immersion a game has over the immersant. While this might be the case, total immersion has been shown to be synonymous with the characteristics of presence. As such, presence might be a better method of gauging the effects games has on a player.

2.6.3 Presence

Presence can be defined as an illusion where the participant believes the environment that they are being presented with, through some medium, is real [21]. It is an important concept for game environments, as high levels of presence can directly attribute to the increase in the amount of enjoyment and fun ascribed to a particular game. Presence has been shown to have a positive affect on the ability to complete tasks in games [35].

There are a number of factors that contribute to presence. These can be loosely grouped as player and media characteristics [25]. Media characteristics refer to elements such as narrative, while player characteristics refer to elements such as the individual's perceptual, motor and cognitive abilities, as well as transient factors, such as mood [25]. Selected and focal attention also plays a contributing role in presence [47], meaning that, for example, with an increase in game difficulty, players might be forced to be more attentive to the game, which gives them less time to focus on the fact that the environment they are playing is not real [35].

Measuring presence

The Independent Television Commision's Sense of Presence Inventory [ITC-SOPI] is a questionnaire used to quantify presence through the use of four independent sub-scales, namely *sense of physical space*, *engagement*, *ecological validity* and *negative effects* [7]. Sense of physical space refers to the feeling of being part of a mediated environment, engagement refers to the emotional aspects that are involved with feeling fun and feeling psychologically involved with the environment, ecological validity refers to the belief that the elements being presented in the environment are real and negative effects refers to the counter-effects experienced in the environment that might negatively influence the degree of presence experienced by the participant [25]. Validations of this questionnaire are promising, as the internal consistency of the test is very high for all four scales ($0.77 < \alpha > 0.94$), meaning that the test is very reliable.

One of the most widely used introspective techniques for measuring presence is the Slater, Usoh and Steed [SUS] questionnaire. The questionnaire consists of six items, which require the participant to asses the level of presence they might have experienced in the game and is administered after they have experienced the game environment. The only downfall of the SUS questionnaire is its subjective nature, which causes internal inconsistencies and validity issues [7].

Apart from using the introspective techniques described above, a number of behavioural and physiological methods are able to measure presence. These assessment techniques rely on physiological responses in order to measure the emotions of the participant. Examples of such techniques include monitoring the heart-rate of the participant. It should be noted that these physiological methods are very difficult to administer and may fall prey to physical inconsistencies, such as reactivity [7].

While a number of techniques have been used to measure presence, no well accepted method has been accepted as the standard measurement for presence [26]. While no consensus has been made as to the most reliable measurement of presence, the ITC-SOPI method has been shown to be the most reliable, and even if not standardised, it is by far the most widely used measure of presence [7].

2.6.4 Flow

Flow can be described as the state where a successful balance between a player's ability and the perceived challenge or difficulty exists [12]. It can be associated with the experience that is in itself so gratifying that people will be willing to do it for its own sake, with little concern for what they might be getting out of it in return or if it was difficult [10]. As such, it is a very important concept to analyse in evaluating games.

According to Csikszentmihalyi [10], flow consists of nine dimensions. These are *challenge-skill balance*, *action-awareness merging*, *clear goals*, *unambiguous feedback*, *concentration on the task at hand*, *sense of control*, *loss of self-consciousness*, *transformation of time* and *autotelic experience*. Challenge-skill balance describes the feeling of balance between the situation and the participants personal skill level. Action-awareness merging describes the level of involvement the participant feels, which might bring upon the sensation that one's actions are automatic. Clear goals describes the feeling of certainty that the participants clearly knows what they have to accomplish. Unambiguous feedback describes the feedback the participant receives, which could lead to a feeling of confirmation as everything might be going according to plan. Concentration on the task at hand describes the feeling of being really focused. Sense of control describes the distinguishing characteristic of being in complete control of the experience. Loss of self-consciousness describes the sensation where all concerns for the self disappears and the participant become fully part of the experience. Transformation of time describes the sensation of time either passing slowly or quickly and autotelic experience describes the end result of flow, where the feeling of doing something is done for its own sake [10].

Measuring flow

Flow has been noticeably difficult to measure, as the only way to directly measure it is by disturbing an already established flow state [26]. Even with this the drawback, several attempts have been made to measure flow.

One of the main areas where flow has been used is in the area of sport. Here, Jackson [20] found support for the nine flow dimensions in a qualitative analysis of elite athletes' flow levels. As such, they developed the 36-item Flow State Scale [FSS], where they used conventional item analysis techniques along with other confirmatory factor analysis in order to establish the construct validity of the scale [20].

Physiological methods, such as facial electromyography [EMG], where changes in facial expressions are analysed, has been used to asses a participant's emotional state [16]. The benefits of using physiological methods, such as EMG, is that it bypasses the need to interrupt a flow state, which so many other methods employ. However, physiological methods of measuring flow are very expensive [7] and allow for other, externally extraneous factors due to the use of external equipment. Because of the external nature of the measurements, it becomes very hard to measure which element of the flow state is triggered and how. That being said, physiological methods are still highly favoured in the research community [26].

One of the first experiments to evaluate flow in games was done by Sweetser and Syeth [42] , where they based their experiment according to the fixed set of heuristics based on the flow dimensions. It was found that their heuristics varied depending on the game genre played and that while heuristics are useful for identifying potential problems, their use becomes unreliable when attempting to extract the emotional effect a environment might have to participants [42].

As mentioned above, flow is hard to measure without breaking the flow state of the participant. While the use of external equipment do bypass this problem, such as EMG, there are still problems in establishing what triggered the flow state. As such, heuristics, while not being able to asses the flow state, do prove useful in identifying which factors contribute to the flow state.

2.7 Summary

Multi-touch is defined as the simultaneous use of two or more fingers as input onto a touch screen, allowing the removal of other external input means. Multi-touch technology was shown to produce a more natural means of interacting with interfaces and does affect the efficiency with which we can accomplish simple tasks. It has been shown that the fastest multi-touch interaction is about twice as fast as a mouse-based selection. Not only does multi-touch interaction "emulate" the way we interact with objects, but it is also a lot more efficient than indirect-touch interactions that we are all used to. That is not to say that multi-touch interactions are without their limitation, as occlusion of the screen objects demonstrates. Support for detecting two fingers will improve performance in multi-touch interfaces, but support for detecting more than two fingers is unnecessary to improve multi-target selection performance. In short, the results are inconclusive. While tests using target selection does analyse basic operations, more advanced interactions, such as the manipulation of three-dimensional objects, needs more analysis. Factors such as learning, practise and muscle-memory also play an important part in future research.

Direct manipulation was defined as the representation of objects and actions, where simplified and reversible actions replace overly complex ones that are continually visible on a interface. While direction manipulation does afford a consistent visual representation of objects, allowing easy learnability and anxious-free interactions for both novice and expert users, it is dependant on the quality of action description and illustration. Gestures are defined as a symbolic method of giving a command to a computer, where its location, size and dynamic properties play a role in its differentiation, and are defined as being a more natural means of interacting with a computing device. Gestures allow for a higher-level of control over commands, allowing for the near removal of interface objects and structures. However, gestures suffer from the lack of feedback and classification, allowing user frustration.

Several approaches have been made in combining gestures within games, including commercial games. However, most research tends to rely on external apparatuses when performing gestures. While gestures do allow for a more natural means of interacting with a game, eliminating the need for intrusive game controllers, they are not without their limitations. Issues such as memorability and gesture distinguishing have negative effects on game-play, especially for highly interactive games which require rapid user interactions. Similarly, games which require external apparatuses face human spatial motion awareness and feedback issues.

The measurement of games proved to be slightly inconclusive, where each component of measurement displayed degrees of inter-connectivity. Immersion was defined as the state in which a immersant feels part of the environment. While different means of measuring immersion does exist, such as the questionnaire developed by Witmer and Singer, it was shown to have no standard means of measurement. However, because total immersion displayed similar characteristics that were present in presence, it might be more appropriate to measure presence in order to measure immersion. Presence was defined as the illusion where a participant believes the reality which they are being presented with is real. Similar to immersion, presence showed a resilience to being accurately measured. While more research has been done in the area of presence and how individuals experience presence, there was no standardised method of measuring it. The only promising measurement of presence was the ITC-SOPI questionnaire. Flow was defined as the state where a successful balance between the player's ability and the perceived challenge or difficulty exists. Most of the means of measuring flow had to break the flow state in order to measure it. The only method that bypassed the interruption of the flow state was with the use of physiological methods, such as EMG. Physiological methods do have drawbacks, such as the expensive nature of the equipment and the lack of identifying which factors are producing a flow state. As such, a combination of physiological and heuristics can prove to be the most useful in measuring flow.

Chapter 3

Gesture Engine

3.1 Introduction

This chapter deals with the specifics of the gesture engine that was developed to facilitate the use of gestures within games. The gesture engine is a crucial part of the project, not only for the development of the game but also as an input mechanism on which this project is based. A breakdown of how this gesture engine relates to the other group members work is discussed, as well as how the gesture engine was designed, implemented and validated.

3.2 Gesture Engine Group breakdown

The gesture engine was broken up into separable components that each group member of the project had to design and implement separately. Nina Schiff worked on the feature extraction, while Daniel Wood and this author worked on different implementations of gesture engines. Where Daniel Wood designed and implemented a gesture engine that used a Hidden Markov Model, Pierre Benz designed and implemented a simple gesture engine based on a state-machine. Figure 3.3 shows the breakdown of components each group member implemented with respect to the gesture engine.

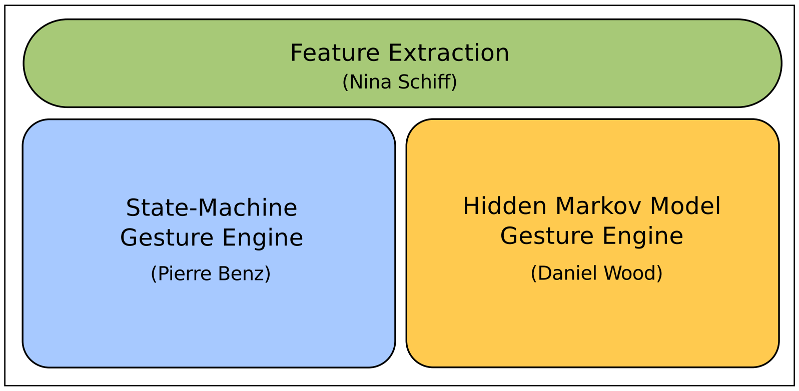

Feature Extraction
(Nina Schiff)
State-Machine Gesture Engine
(Pierre Benz)
Hidden Markov Model Gesture Engine
(Daniel Wood)

Figure 3.1: An overview of what each group participant contributed to the gesture engine.

3.3 Motivation

The motivation behind the design of the gesture engine presented in this report is threefold. Firstly, the engine is designed to be *real-time*. Traditionally, gesture recognition is processed after all input has been specified [44], which works well if one has computational time to spare between user input and computational output. This, however, is not the situation when it comes to games, as user input is only one among many other computational operations needed to be done in real-time. The gesture engine should recognise features as they are processed, which not only alerts the user if they are performing the correct gesture as they perform it, but also ensures that the user is not surprised in case they performed an incorrect gesture.

Secondly, the engine needs to be fast and light on resources. Since the games are designed to run on the Apple iPad, which is less powerful than standard gaming devices available, the recognition of gestures should not take up all the resources of the device.

Lastly, the engine should allow for the simple and easy creation of new gestures. This is essential, as group members will not have time to sit and experiment with input values in order for them to use gestures in their games.

3.4 Feature Extraction

In order to differentiate each gesture from one another, the symbol inputted by the user needs to be broken up into a pattern that accurately represents the gesture. Because the gesture engine's main function is to distinguish and identify each feature as it arrives from the feature extractor, it means that the development of the gesture engine had a dependency on the feature extraction implementation. Major changes in the way each feature was broken up and represented meant that the underlying gesture engine had to be able to interpret it.

Two different feature extraction methods are used. The *Linear Node* gesture engine uses a parabolic curve-fitting feature extraction method, while the *State Machine* and *Grouped Node State Machine* gesture engine uses a change-in-direction feature extraction method.

The design and implementation of the different feature extraction methods can be found in Nina Schiff's report [39], as well as the limitations of each method. In order to understand the design decisions and motivations behind the gesture engine described here, a brief summary of the feature extractor limitations is presented. While the feature extraction that uses a parabolic curve fitting method is able to handle curved gestures reasonably successfully, it struggles to handle downward vertical lines. This causes a number of otherwise distinguishable gesture features to be dropped and seriously affects the implementation of the *Linear Node* gesture engine, as it can not consistently handle gestures that incorporated downward vertical lines as a distinguishing feature. The feature extractor which uses a *change-in-direction* method successfully handles all gestures, but struggles to handle curved gestures consistently. Figure 3.2 shows the two different feature extraction methods used in the development of the gesture engine, as well as how both feature extraction methods handled certain gestures.

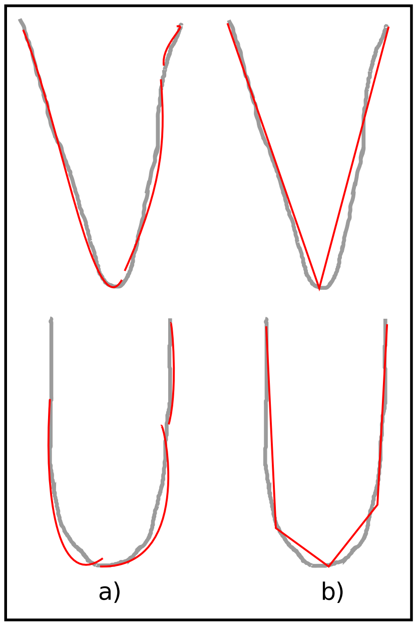


Figure 3.2: An example of two different feature extraction methods used in the gesture engine. Both feature extractions are performed on 'V' and 'U' gestures. a) shows the use of parabolic curve-fitting feature extraction, while b) shows the use of the change-in-direction feature extraction. Note that the parabolic curve-fitting does not recognise the initial vertical line drawn, which often resulted in 'U' gestures having the same features as 'V' gestures.

3.5 Linear Node

3.5.1 Design

The first implementation of the gesture engine took the form of linear nodes. Here, the main implementation consists of a large array containing gestures definitions. Each gesture type is defined individually and corresponds to each feature sent from the feature extractor. Gesture types do not share nodes with any other nodes in the engine, which means that each node only has one parent and one child. Once a node is found, moving to its child was just a matter of correctly keeping track where the gesture is in the engine.

Each gesture node is composed of a values which help distinguish it from other features. Firstly, it is composed of a `a`, `b` and `c` value, where each value corresponds to the parabolic equation:

$$y = ax^2 + bx + c \tag{3.1}$$

These `a`, `b` and `c` values are the mean values derived by a process of performing a gesture 100 times, which was performed by Nina Schiff. Because every person performs a gesture differently, a rough average of the gesture would provide the general case. This is essential in the gesture recognition engine, as the classification and recognition of gesture features enables gestures to be distinguished from one another. This means that even though the values computed give a rough estimation of the values of the feature, it is still liable to fail. As such, error values corresponding to each of the parabolic equation variables are also added to the gesture node. Lastly, each gesture node is composed of an end point value in integer Lpez-Dahab [LD] coordinates, which allows for a calculation of the direction of gesture motion.

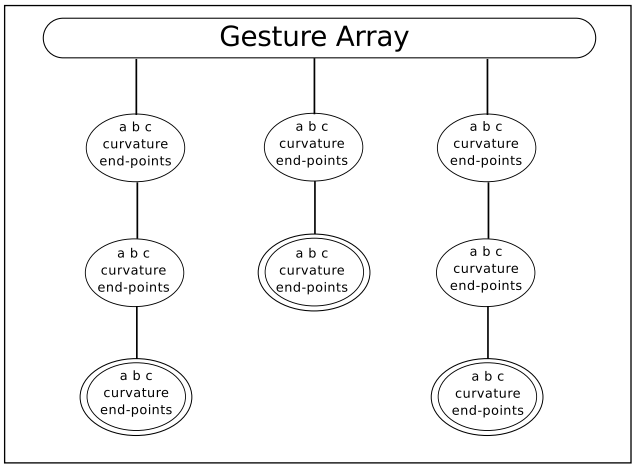


Figure 3.3: An example of the *Linear Node* gesture engine, where each parent node has only one child

There are two modes in which the gesture engine is used. Initially, the gesture engine loads gesture definitions from a file. This allows developers to specify which gestures are able to be recognised by the engine and what their features should be. A file format is used to specify each gesture and is loaded into the gesture engine at the gesture initialisation. The file format is as follows:

```
gesture identifier  (string)
number of features  (integer)
...
a b c               (float)
a b c error values  (float)
end points          (integer)
...
```

In the second mode of the gesture engine, data is received from the feature extractor. This occurs at run-time. When the feature extractor detects a feature, a call to the gesture engine is made, specifying the feature's parabola and end-point. The gesture engine then checks to see if the values specified from the feature extractor can be found in the gesture engine.

3.5.2 Implementation

The *Linear Node* gesture engine is the first iteration of the gesture engine. It consists of a *Node* and an *Engine* class. Gesture definitions are implemented by reading in files at the *Engine* object creation, at which point files are parsed and loaded into the engine. These nodes where packed into a two-dimensional `NSMutableArray`. The engine has three additional methods, namely `StartGestureRecogniser`, `EndGestureRecogniser` and `doesGestureExist`, which perform the actual recognition methods in conjunction with the feature extractor.

The `StartGestureRecogniser` method is an initialisation method for the gesture recogniser engine and marks the start of the recognition process. Additionally, it resets all tracking variables.

The `doesGestureExist` method is called every time a feature is detected. It serves as the main function for real-time gesture recognising, as it updates the gesture engine according to how far along the gesture nodes the feature has progressed. Once the method has been called, a number of steps are taken. Firstly, an iteration through the gesture array is undertaken if no previous features have been sent to the gesture recogniser. Here, every single starting node has its node values checked against the values sent to the method. More specifically, a check is made to see if the gesture direction and the input directions are aligned. This determines if the direction is either going in the positive or negative x and y direction. A simplification is imposed, in that the gesture node's x and y end-point values are rounded to either 1 or -1. The value is 1 if the value is positive and -1 if it is negative. A similar method is applied to the end-points passed to the method. The values are multiplied, and if they are in the same direction, the value should be equal to 1. If this check has been successfully completely, the parabolic equation values are checked against the node's parabolic equation values, padded with the parabolic error values. If the inputted parabolic equation values lie within the padded parabolic values, then the node is recognised and tracker variables mark that node. The pseudo-code for the function is as follows:

```
doesGestureExist()
    loop through gesture array
        if features are in the same direction
            if (a, b, c) values lie in indexed node's padded values
                track this node
```

The `EndGestureRecogniser` method is the final stage in the gesture recognition engine and is called once all feature extractions have been completed for a gesture. This is also the end stage in the gesture recognition process, meaning that if the state is in a final *Node* state, the a complete gesture has been recognise. Otherwise, a complete gesture has not been recognised, even though past nodes might have been recognised in the process.

3.5.3 Limitations

The *Linear Node* gesture engine has many limitations and it was decided to modify the way in which the gesture engine handles gesture internally and externally. Firstly, the use of a gesture array has certain performance issues. Continuous iterations through the gesture array and the use of tracker variables in order to keep track of which index of the array is successfully being recognised, not only makes for complicated code but also seriously affects the rate at which other operations are executed. For example, if sixteen gestures were loaded into the gesture engine, the gesture array would be iterated sixteen times, regardless of which features have previously been recognised. Secondly, because each state is independent of the whole engine, it means that nodes sharing the same values or multiple nodes recognised by one inputted gesture are all equally valid, and no differentiation is possible as to which gesture is actually being recognised. Similarly, because each node is independent of other nodes, it means that the gesture engine grows significantly with the number of gestures. Lastly, classifying gestures by hand is a painstaking task, as numerous trial-and-error modifications need to be made to the parabolic error values to ensure that the correct gestures can be recognised. Even if they are recognised, this recognition only applies to the person testing and adjusting the gestures, as they are suited to their method of drawing and are not necessarily to other users.

3.6 State Machine

3.6.1 Design

Addressing the issues of the *Linear Node* gesture engine, a directed-graph state-machine is proposed as a solution. Instead of using an array of gestures, where each node is independent of all the other nodes in the engine, a *StateMachineNode* is designed to have multiple parents and children. This means that only one node can be accepted in the sequence of gesture features being inputted, which eliminates the need for tracker variables and any post-gesture recognition methods used for distinguishing completed gestures.

It should also be noted that several adjustments are required for the feature extraction. A change-in-direction method of feature extraction was selected as the best method of extracting features from gestures being performed. As such, parabolic equation variables, error parabolic variables and end-point variables are not being used to distinguish gestures from one another. Instead, a gesture direction map is used (as can be seen in figure 3.4). The direction map converts features to a 360° Cartesian plane and specifies features in increments of 0, 45, 90, 135, 180, 225, 270 and 315°. This was found to adequately describe all features coming from the feature extractor. *StateMachineNodes* are described in directions, which means that the total number of gestures nodes remain small as the nodes are described in a manner which explains overlaps.

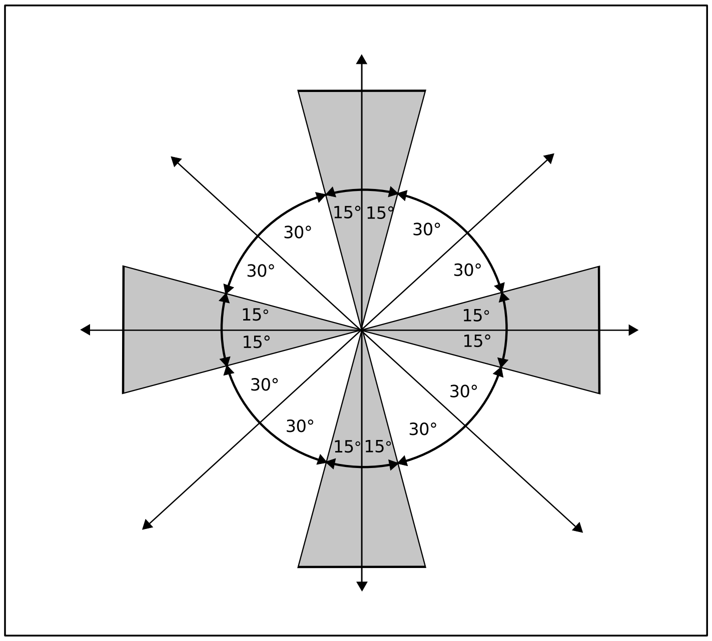


Figure 3.4: The direction map the *State Machine* gesture engine uses to map features. Also shown is the zones in which features are rounded.

3.6.2 Implementation

The implementation follows a similar structure to the *Linear Node* gesture engine, in that it reads the input file at creation and contains `StartGestureRecogniser`, `EndGestureRecogniser` and `doesGestureExist` methods. This was meant to keep the gesture engine Application Programming Interface [API] stable, so that different back-ends could be swapped engine-side without impacting other areas and other group members.

Gesture definitions are read from a file at the *Engine* object creation, where the files are parsed and loaded into the engine. A root node, constructed from a *StateMachineNode*, is the highest-level node of which first level gesture definition nodes are children. As the parser goes through all the input files, it first checks if a node already exists its children, in which case it does not create a new node but merely sets it as its current location and sets its parent node as the found node's parent. If the node does not exist in the gesture engine at that particular level, then a new *StateMachineNode* is created and set as the the current nodes child. In this way, node duplication is avoided. Once all the input files have been loaded into the gesture engine, the engine is ready to be used by the feature extractor.

The main difference between the `doesGestureExist` method here as compared to the *Linear Node* engine, is the removal of gesture tracker variables and unnecessary iterations through the gesture array. A current gesture node is kept in memory, and checks are made against the current node's children to see if the feature being passed is accepted by one of the children. As the children will be different from one another, only one child can be accepted at a particular node. Within node checking, all parabolic parameter are removed. Instead, the feature's rounded direction is checked. This means that directions which fall in the 90° range (0°, 90°, 180° and 270°) will accept features with angles of 15° of difference. For example, if a feature has an angle of 81°, then it will be rounded to 90°, since it falls in the range of (75°, 105°). Similarly, angles at 45° will be accept features of 30° difference. Once the direction has been discretized, the children are checked to see if the feature's direction matches that of the current gesture node's child. If a child node does exist, then the current node is set to that child node, marking that the feature is accepted. Otherwise, the gesture engine signals to the feature extractor that no matches have been found. The pseudo-code for this function is as follows:

```
doesGestureExist()
    select current node's children
        if features are in the same direction as child node
            set child node to current
        else
            signal to feature extractor that no matches have been found
```

3.6.3 Limitations

Several limitations arose while implementing the gesture engine using the *State Machine* approach. Firstly, the size of the gesture engine remains significant. Even though the engine fares better at recognising gestures than the *Linear Node* gesture engine, and at times it performs with a 100% recognition rate, multiple gesture definitions still need to be loaded into the gesture engine in order to recognise similar gestures. This is especially the case when a curved gesture is being specified. An example of this is the *spiral* gesture, composed of curves and not straight lines. This means that about 8 or more different gestures files need to be loaded into the gesture engine in order for it to recognise every different gesture drawing case, not only making it cumbersome to define the gesture but also in loading to a large gesture file. Figure 3.5 demonstrates the issues with the spiral gesture, where the feature extractor successfully handles all spiral features but makes it almost impossible to handle all feature cases in a single gesture file. The gesture engine needs to be modified to handle these curved gestures more appropriately.

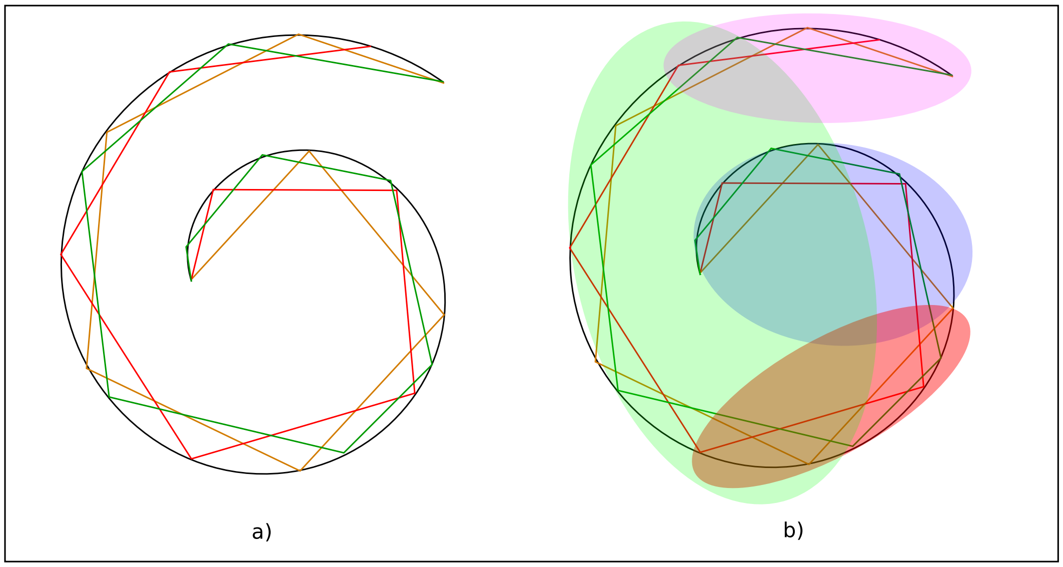


Figure 3.5: An example of a spiral gesture being performed with feature extraction displayed as coloured lines. a) multiple ways of extracting features exist, which shows the trouble in easily distinguishing a spiral gesture. B) displays a solution the *Grouped Node State Machine* uses to address the curved features present in the spiral gesture.

3.7 Grouped Node State Machine

3.7.1 Design

The *Grouped Node State Machine* is designed to address the issue of curved features in inputted gestures. As figure 3.5 demonstrates, features with curves requires multiple gesture definition cases. One way of addressing the problem is by grouping curved features into a "special" node, which allows permutations without creating multiple gesture definitions for a gesture. An example can be seen in figure 3.5(b), where curved features are grouped, as well as figure 3.6, where the gesture engine for the states of a spiral is shown, comparing the *State Machine* and *Grouped Node State Machine*.

A further requirement at this juncture is that the engine support multi-touch. While the multi-touch handling is done in part by the feature extraction process, the internal gesture states needs to differentiate one-finger gestures from multi-finger gestures. As such, the *StateMachineNode* class has to include a multi-touch value, which means that multi-touch gesture nodes distinguished themselves not only in direction, but also in number of touches.

Furthermore, the input file for handling gesture definitions is to be modified to handle not only multi-touch, but also grouping of directions in order to handle curved features. As such, the file has the following structure:

```
gesture identifier (string)
number of features (integer)
```

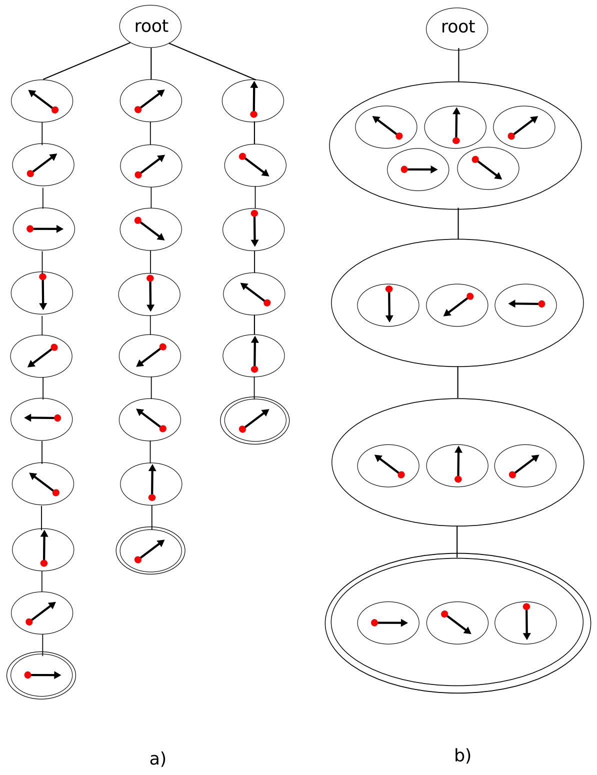


Figure 3.6: An example of the gesture nodes present in both the *State Machine* and *Grouped Node State Machine* for a spiral gesture. As can be seen, a) has multiple gesture definitions to a handle a single gesture, whereas b) shows a single gesture definition which handles all possible variations.

```
..
number of touches  (integer)
nodes grouped      (integer)
angles in groups   (integer)
..
```

3.7.2 Implementation

The implementation has the same structure as the *State Machine* gesture engine, with the modification that the *StateMachineNode* allows the inclusion of a "grouped" node, which allows a node to have multiple directions, and an inclusion of a touches variable, that specifies how many fingers are performing a gesture.

The `doesGestureExist` method is modified to incorporate multiple touches from the feature extractor, as well as handle multiple directions within each node. Firstly, a check is made against the number of fingers used by the feature extractor. This is a quick pass-fail test, which means that the gesture engine would only have to check against similar numbered touch nodes. If the number of fingers used by the feature extractor matches the number listed in a node, then the gesture engine performs actual feature comparisons. In order to accomplish the inclusion of grouped nodes within a gesture, several checks are made. Firstly, a check is made against the current node's directions. If the current node has a direction that is successfully checked against the inputted feature, then the node pointer stays on the current node. If the feature's direction value is not found in the current node, then a second check is made against the current node's children and their direction values. Figure 3.7 shows a simplified example of this process. The direction checks are exactly the same as the *State Machine* gesture engine, where the feature extractor's directions are rounded to the nearest 45° value.

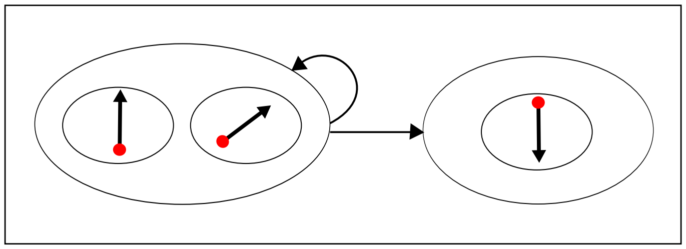


Figure 3.7: An example of two *StateMachineNodes* which are used in the *Grouped Node State Machine*, where a check is made against the current node's directional values, after which a check is made against the child node.

3.7.3 Limitations

While the *Grouped Node State Machine* implementation successfully handles spirals and other curved gestures, it still suffers from limitations. While grouping of direction values within a "grouped" node does help simplify the detection and creation of gesture definitions, it also overly complicates the gesture engine. Careful design of gestures is need to address each state, ensuring that each gesture can be successfully found and does not conflict with other gesture nodes. As such, many gestures needed to be modified to address the inclusion of grouped nodes. However, several problems arise. If a grouped node successfully "swallows" other, lone-standing nodes, and they are gesture endings, which one of the gesture endings did the gesture engine find? Luckily, in the gestures designed in this project, no gestures ever displays this problem.

3.8 Validation

In order to test how well the gesture engine mentioned above performs in comparison to a probabilistic gesture engine based on a Hidden Markov Model (as developed by Daniel Wood [49]), several tests were done to test how accurate the engine was and to test for the occurrence for type 1 and type 2 errors. Type 2 errors are described as "false negative", which, in the context of a gesture engine comparison, means that a gesture is falsely recognised by the gesture engine even though it was correctly inputted into the device. A type 1 error is know as a "false positive", which would be when a gesture engine recognises the gesture but classifies it as a different gesture to the one being inputted.

The test was performed by using the different engines, both loaded with 'V', 'U', 'Z' and 'spiral' gestures. Each gesture was performed a hundred times by Daniel Wood and the author. The results of these tests can be found in table 3.1. T-tests were performed on the results provided by the gesture comparisons and can be found in table 3.2. The analysis of the data shows that there is no significant difference in the scores produces by the *Grouped Node State Machine* and the *Hidden Markov Model* gesture engines for correctly finding the gestures with conditions t(3) = -0.2645, p = 0.8085. Similarly, no significant differences were present in Type 1 errors with conditions t(3) = 1.8446, p = 0.1623, nor with type 2 errors with conditions t(3) = -0.7108, p = -0.5285. These results suggest that both engines performed similarly and that the choice of gesture engine should not be based on accuracy.

		Gestures			
		V	U	Z	Spiral
Grouped Node State Machine	Correctly Classified	86%	74%	97%	99%
	False positives	14%	11%	0%	1%
	False negatives	0%	15%	3%	0%
Hidden Markov Model	Correctly Classified	87%	95%	99%	83%
	False positives	0%	0%	0%	0%
	False negatives	13%	5%	1%	17%

Table 3.1: Results from gesture engine comparison. Percentages of classification in different categories.

	mean of differences	t-value	df-value	p-value	95% confidence value
Correctly Classified	-2	-0.265	3	0.809	-26.062 22.062
Type 1 Errors	6.5	1.845	3	0.162	-4.714 17.714
Type 2 Errors	-4.5	-0.711	3	0.53	-24.649 15.650

Table 3.2: Statistical T-test results performed on the gesture engine comparisons.

3.9 Future Improvements

There are several improvements that could be made to the gesture engine described in this report. Firstly, a more robust mechanism for handling complex gestures should be incorporated into the gesture engine. This includes features such as gesture-chaining, where several sub-gestures can be executed sequentially. A more elegant method of handling complex multi-touch gestures needs to be developed, as the current gesture engine only handles multi-touch gestures where the number of fingers remain constant during the duration of the gesture. Lastly, the gesture engine is tightly coupled to the feature extraction for efficiency reasons. If the feature extraction had to change to incorporate curvature more accurately, then the gesture engine would need to support it.

3.10 Conclusion

In conclusion, a gesture engine has been developed to handle features extracted in the form of a state-machine. The state machine has multiple implementations, with the final implementation handling multi-touch gestures in the form of state nodes. Certain nodes are grouped in order to facilitate curved gestures, which can be extracted in a variety of ways. It is also shown that the gesture engine performs comparable to a probabilistic gesture engine, in the form of a Hidden Markov Model, in terms of differentiating and accepting gestures. As such, the gesture engine presented here is adequate to being used for all the games used in this project.

Chapter 4

Game design

4.1 Introduction

This chapter describes the game that was designed to investigate the usage of gestures in games. The game is one among the three games which is used in the user experiments. A brief introduction on the game genre that the game was based on is described, as well as the design and implementation of the game, followed by a short test study in which the game was validated. The reasons for the game designs presented here are twofold, namely that they describe the internal mechanisms of actual game-play, mentioning and providing motivations for why certain game mechanics are used, and how they are incorporated within the wider framework of addressing the research questions pertaining to the effects of gestures in games and game gestures.

4.2 Strategy game

The game genre the game is loosely based on is a strategy game. The main emphasis in strategy games is the use of strategic planning and tactics in order to accomplish certain goals. The player is generally in control of a certain amount of units, which they can either build or control, as well as assign commands to. The goals of the game vary from eliminating all opposing units to capturing targets. Most strategy games involves a certain amount of warfare, as well as the use of environment exploration and resource management.

Strategy game are usually broken up into two groups, namely real-time strategy and turn-based strategy. Real-time refers to continuous game-play, where all actions are done in real time, whereas turn-based refers to the use of discrete, turn-based strategy, where each side has a phase to perform actions.

4.3 Game design

4.3.1 Game-play and key features

The game developed, *Capture The Towers*, is a real-time strategy game, where the objective is to capture three enemy towers, which are surrounded by enemy units. The player controls a number of units, which are used to accomplish the objectives of the game.

World viewing

The player has the ability to view the world environment, which is presented in a top-down, map-like overview. This allows the player to see the terrain, friendly and enemy units. Moving around the map is performed by either using the map-navigational buttons, which is only available in the direct manipulation version of the game, or using map-movement gestures, which is performed by performing a two-fingered swipe gesture.

Resources

The player starts the game with five hundred resources, that can be used to build units. Every time a tower is successfully captured, the player will gain two hundred and fifty additional resources, which can be used to build new units. Table 4.1 displays the resources required to build a specific unit. These resources are constantly visible to the player and is situated at the top of the screen, next to the remaining time left to complete the game.

Unit name	Cost
Foot soldier	20
Archer	30
Chaplain	80
Trebuchet	250

Table 4.1: The cost of each unit available in the game

Time-limitation

In order to successfully complete the game, all three enemy towers need to be captured within 7 minutes of game-play. The player is unsuccessful at accomplishing the game requirements when this is not done, which results in the game ending. The time allocated, and the remaining time left for the player to accomplish the game objectives, is visually displayed to the player through the use of a horizontal bar, which appears in the top-centre of the screen.

The enemy

Enemy units occupy the area surrounding the towers that the player has to capture. They remain stationary and do not move from their allocated positions. They will automatically resort to combat if any player units come within their interaction-circle.

The towers

Capturing the enemy towers is the main objective of the game. There are a total of three towers, which are located across the map. An assorted number of enemy units surround each tower, which the player will have to eliminate before they can capture the tower. Capturing a tower is done by positioning a unit over the tower for an allocated time of 20 seconds and can only be performed by foot-soldiers or archers. When the capturing process has been started, visual indications of the capturing process will be displayed with a pulsing animation. Once a tower has been captured, the tower will change to a blue colour, indicating that it now belongs to the

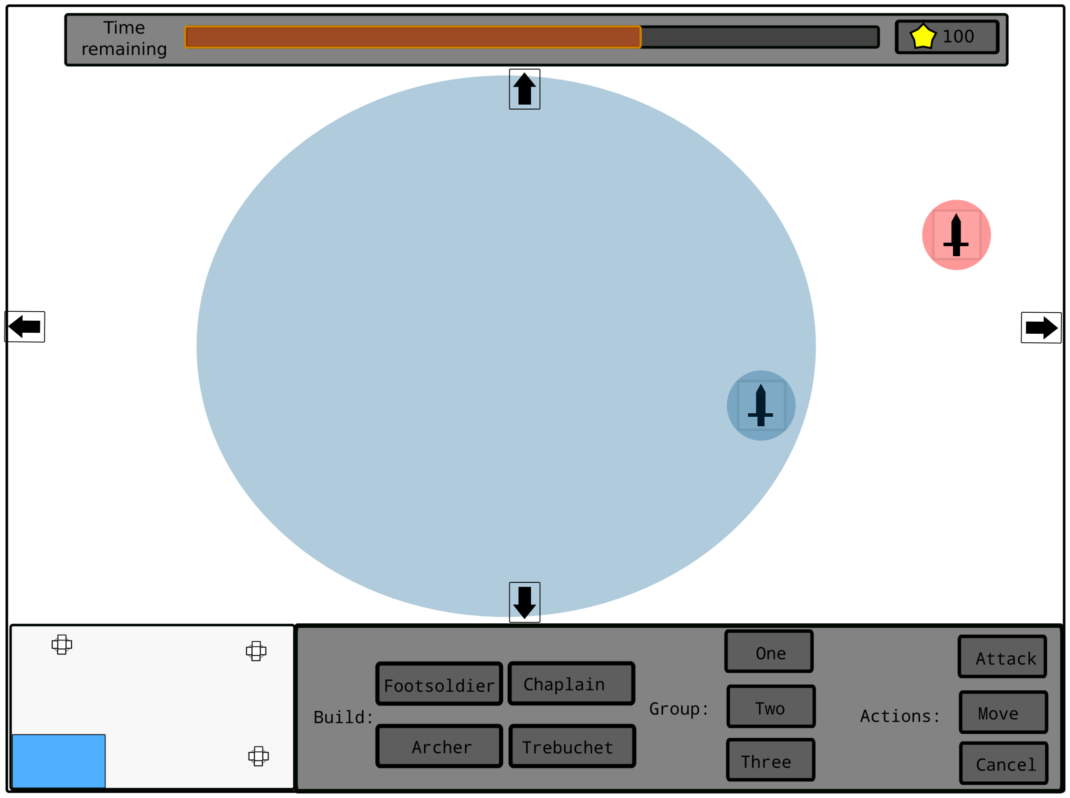


Figure 4.1: An example of the direct manipulation interface showing the placement area after a player selected a foot-soldier to be built

player. Additionally, the player will receive 250 resources, which can be used build additional units.

Building units

In order to add player controllable units in the game environment, the player will have to build units. This becomes especially useful when the player has captured enemy towers and wants to use the resources gained to build more units. Building a specific unit is done by selecting a unit from the user interface or, if the player is playing the gesture-based interface, performing a unit gesture corresponding to the the unit they want to build. Once a unit has been chosen to build, an area of placement is presented to the player. This area is located at the player's initial starting position and is visually indicated by a blue circle. If the player has moved away from this location, for example, by moving the screen to a different area of the game map, the screen will animate back into this initial location and will allow the player to place units in the game map. Interacting with the placement area is done by tapping within the circle presented. If the player does not tap within the circle, the building action is cancelled and the player will have to redo the unit building selection process.

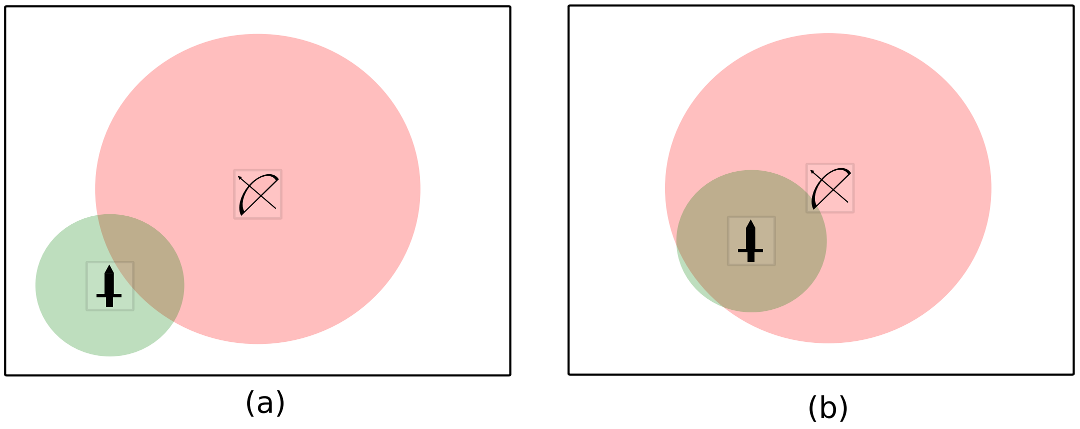


Figure 4.2: Interaction circles of two units, where (a) shows a foot-soldier within the interaction-circle of a archer, causing the the archer to attack the foot-soldier, and (b) shows a foot-soldier and archer within each others interaction-circles, where both units will attack each other

Unit name	Interaction-circle radius
Foot soldier	20
Archer	200
Chaplain	150
Trebuchet	400

Table 4.2: Unit Interaction-circle radii

Combat

Combat happens whenever a unit occupies the interaction-circles of an opposing unit. Certain units, like archers, can attack from a distance, whereas foot-soldiers need to be at a closer distance from their opponents in order to attack them. Interaction-circles differ among unit-types, and table 4.2 shows the relative interaction-circle each unit will have. Figure 4.2 shows an example of unit interaction-circles that will be used in the game.

When a unit occupies the interaction-circle of an opposing unit, the unit whose interaction-circle has been infiltrated will target the infiltrating unit. A unit can only target one unit at a time and will continue to attack that unit unless they move out of the targeted unit's interaction-circle's range, if they or the opposing unit die or if the player assigns the unit to attack a different target.

Once a unit is targeting an enemy unit, they will start attacking. Attacking is performed with an attack-speed time delay. This means that units will only be able to perform an attack on their targets once they have waited an specific amount of time before performing the next attack. Once a attack has been made, a secondary check is made to ensure that the attack is successful. This check is performed by generating a random number between the values of 0 and 20, which will be the attack value. If this attack value is greater than the targeted unit's defence modifier, then a successful attack is made. If this value is not greater than the targeted unit's defence modifier, then the attack is unsuccessful. If the attack is successful, then the attacking unit will deal damage to the unit being attacked. When a unit takes damage from an attack, its hit-points will decrease. Once a unit's hit-points reach 0, the unit is classified as dead and is removed from the game map.

Visual indication is given to units attacking targeted units. In order to display that the unit is attacking a target, a red line from the attacking unit to the targeted unit is drawn. In order to animate the attack speeds of each unit and to differentiate attacking units from normal, stationary units, a pulsing animation is applied to an attacking unit. This is done to indicate when when each attack will take place. This pulsing animation takes the shape of a circle and fills the whole area of the unit. A special note should be made about the chaplain. The chaplain is a unit that does not deal any damage to enemy units and only heals friendly units. A chaplain can only heal friendly units if they are within the chaplain's interaction-circle. Once there, the chaplain will heal them every 10 seconds. Healed units will gain an additional 5 hit-points every time they are healed by a chaplain. Chaplain's also displays a healing pulse radius, which indicates when a healing action will take place.

If a unit fires a projectile, like the archer or the trebuchet, the unit does not deal direct damage. Instead,a projectile is launched at the position where the targeted unit was when the attack was made. Projectiles only initiate in a attack sequence if the projectile collides with an enemy unit. This also means that a unit could outmanoeuvre a projectile aimed at them. Projectiles only collide with enemy units and will not deal any damage to friendly units. Any enemy unit that intercepts an oncoming projectile will collide with the projectile, causing the projectile to initiate a attack check and destroy itself. Once a projectile has collided with a target, an attack check is made. This is similar to the attack sequence mentioned above. If the attack value is above the targeted unit's defence modifier, then a successful attack has been made and projectile damage will be applied to the targeted unit. A note should be made about the trebuchet projectile type. The trebuchet fires a boulder at a location. This boulder is projected into the air and can not be intercepted by other enemy units. Once it reaches its target, it deals an area of damage, where any enemy unit that is located in the area of damage starts an attack check. If the attack check is unsuccessful, meaning that the attack value is greater than the unit's defence modifier, the unit is hit and takes damage. Table 4.3 displays the different projectiles, with their movement speed, damage and their radius of effect.

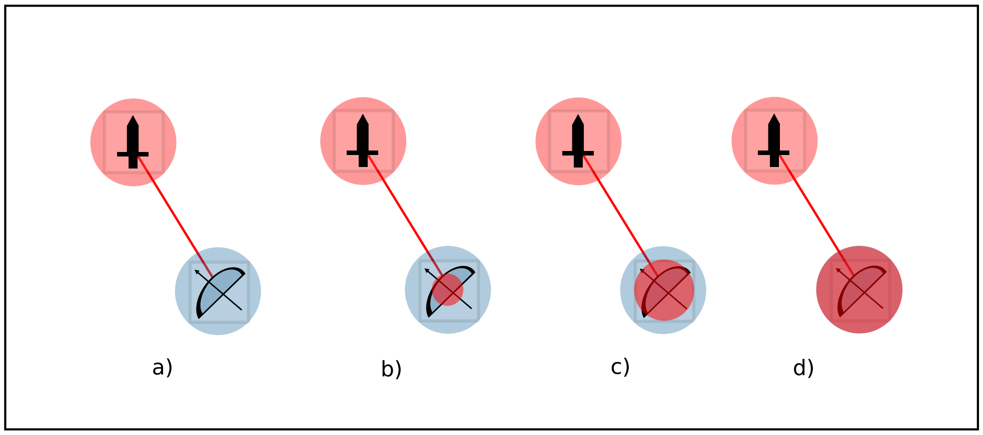


Figure 4.3: An example of an attack sequence. A red line is drawn between the archer and the foot-soldier, with a pulsing animation shown in the consecutive images. In step d) the archer has waited sufficient amount of time to perform an attack check, where in steps a, b and c the red circle shows how much the archer still has to wait in order to perform an attack.

Projectile origin	Movement speed	Damage	radius of effect
Archer	7	2	-
Trebuchet	3	20	80

Table 4.3: A summary of the projectiles associated with units and their specifications.

Unit name	Hit-Points	Attack damage	Defence modifier	Attack speed	Movement speed
Foot soldier	50	5	5	10	4
Archer	20	see table 4.3	10	50	3
Chaplain	15	-	-	-	2
Trebuchet	100	see table 4.3	1	100	1

Table 4.4: A summary of the units available and their specifications.

Table 4.4 displays the unit specifications of all the units available in the game, including unit hit-points, attack damage, defence modifier, attack speed and movement speed.

Artificial intelligence

The enemy will have very basic artificial intelligence. No attacking will be done unless the player attacks a specific unit or if an enemy unit comes into its interaction-circle. Units will stop attacking once the opposing unit is out of its interaction-circle.

4.3.2 Units

Types of units

The game features a range of units that the player will be able to control. Each unit has their own strengths, weaknesses and abilities. A summary of units available is displayed in Table 4.5.

Foot-soldier The foot-soldier is the standard unit of the game. It has the strongest defence in the game, but comes at the price of having a short attack range, since it can only attack enemy units that stand next to it.

Archer The archer provides the standard long-range support in the game. Being able to attack from a long distance and having a fast attack speed, the archer is great for providing covering fire for units.

Chaplain The chaplain is a specialist unit, which functions as a healer. The chaplain can heal nearby units for a certain amount of time, after which it will have to recharge its ability. The chaplain has the weakest defence and the can not attack enemy units. The chaplain can only heal surrounding units once it is stationary.

Trebuchet The Trebuchet is a siege weapon which has the longest attack range of all the units available. While it can deal a lot of damage if it hits a unit, the weapon is very inaccurate. As such, contrasted to other units who can only attack individual units on a one-on-one basis, the trebuchet attacks a certain area, which has a radius of damage. However, the projectile fired

Image	Name	Characteristics
🗡	Foot soldier	Short attack distance Hand-to-hand combat Normal attack speed Strong defence Fast movement
🏹	Archer	Long attack distance Arrows (projective) Fast attack speed Weak defence Normal movement
✚	Chaplain	Ability to heal nearby units Unable to attack Weak defence Slow movement
●	Trebuchet	Great attack distance Siege weapon (projectile) Slowest attack speed Weakest defence Slowest movement No close combat

Table 4.5: List of units available in the game

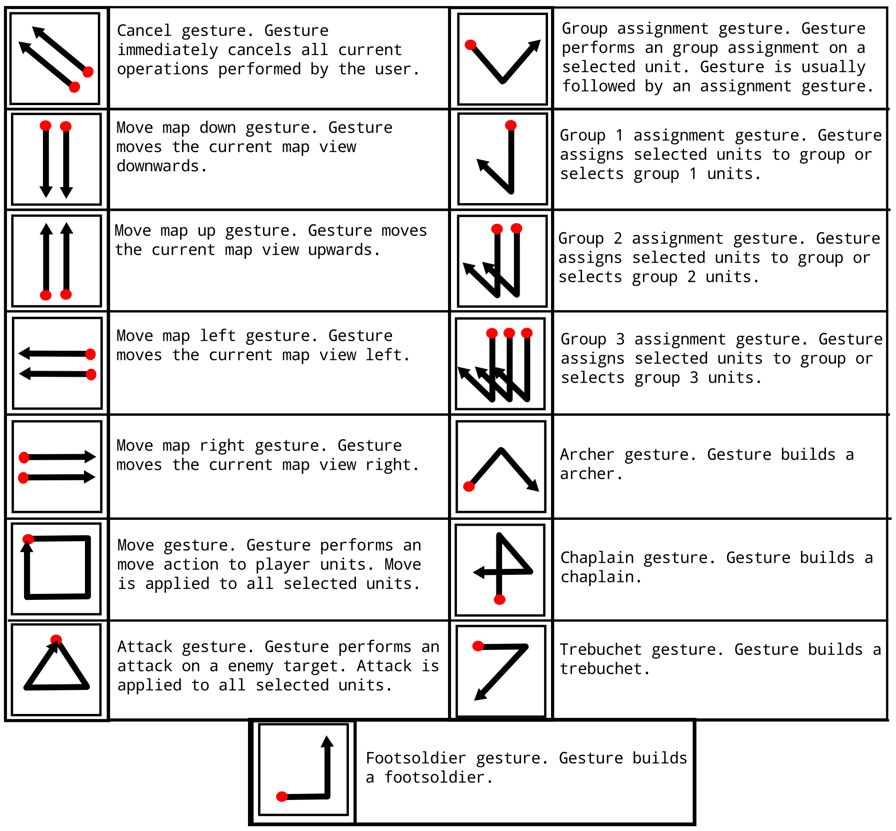
from the trebuchet is slow moving, meaning that units might be able to outmanoeuvre it. The trebuchet is the slowest moving unit in the game and takes the longest time to recharge between attacks. It also has a fixed attack range and cannot attack any units at a short distance.

Selection and commands

In order to interact with a specific unit, the player simply selects the unit by tapping directly on it. Once the unit is selected, the player can issue unit specific commands. Basic commands include moving to a specific location on the map or attacking an enemy unit. If an unit is selected, a light-blue selection circle will be drawn around it to indicate that it has been selected by the player.

While the player could continually control and command all their units, units can survive in the game world without any player interaction. For example, if an enemy unit comes within the range of a unit's interaction-circle, it will automatically attack the unit.

4.3.3 Gesture design



	Cancel gesture. Gesture immediately cancels all current operations performed by the user.		Group assignment gesture. Gesture performs an group assignment on a selected unit. Gesture is usually followed by an assignment gesture.
	Move map down gesture. Gesture moves the current map view downwards.		Group 1 assignment gesture. Gesture assigns selected units to group or selects group 1 units.
	Move map up gesture. Gesture moves the current map view upwards.		Group 2 assignment gesture. Gesture assigns selected units to group or selects group 2 units.
	Move map left gesture. Gesture moves the current map view left.		Group 3 assignment gesture. Gesture assigns selected units to group or selects group 3 units.
	Move map right gesture. Gesture moves the current map view right.		Archer gesture. Gesture builds a archer.
	Move gesture. Gesture performs an move action to player units. Move is applied to all selected units.		Chaplain gesture. Gesture builds a chaplain.
	Attack gesture. Gesture performs an attack on a enemy target. Attack is applied to all selected units.		Trebuchet gesture. Gesture builds a trebuchet.
	Footsoldier gesture. Gesture builds a footsoldier.		

Figure 4.4: List of gesture available in the gesture-based and mixed game interface

Gestures allow the player to perform to interact and control the game with the use of gestures. Figure 4.4 displays a list of gesture available in the game and are also displayed in the game.

General Purpose Gestures

In case the player makes a mistake or wants to perform a different action than the one currently being performed, a general cancel gesture is available.

Map Gestures

Other than directly manipulating the world through touch events, the player also has access to specific map gestures that manipulate the way the player views the world.

Unit Grouping Gestures

Units have specific grouping gestures which can be applied to them by the player. This allows for more high-level of control and allows the player to command more than one unit at a time.

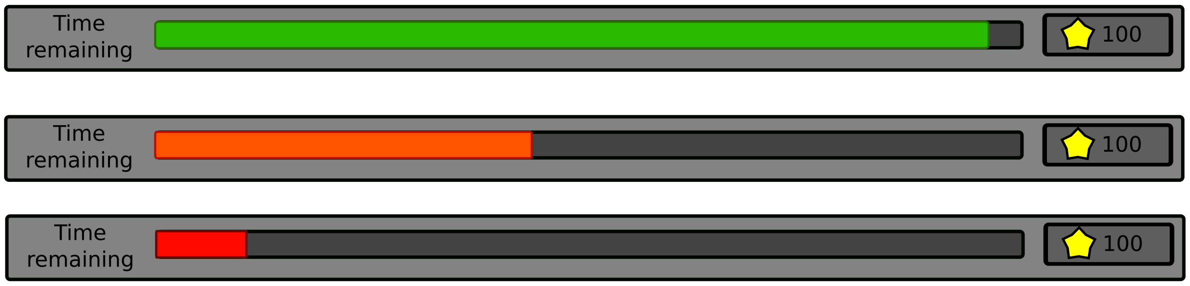


Figure 4.5: Time-limitation interface, where a coloured bar indicates the remaining time the player has to complete the game and the severity of the total time left

4.4 User interface

4.4.1 Time-limitation interface

The time-limitation interface is constantly visible to the player. The bar is colour coded, indicating the total time left for the player to complete the game. Figure 4.5 shows the time-limitation interface and the different colour states it will be in.

4.4.2 Unit interface

The unit-interface is located at the bottom of the screen and displays the unit currently being selected by the player. Three different interfaces are designed to accommodate the direct-manipulation, mixed and gesture-based interface. With regards to the selection of unit modifiers, the gesture-based interface will rely on gestures to modify the unit abilities, while the direct-manipulation interface will require the player to directly interact with the user interface elements displayed at the bottom of the screen. This mixed interface will combine a combination of direct manipulation and gesture objects.

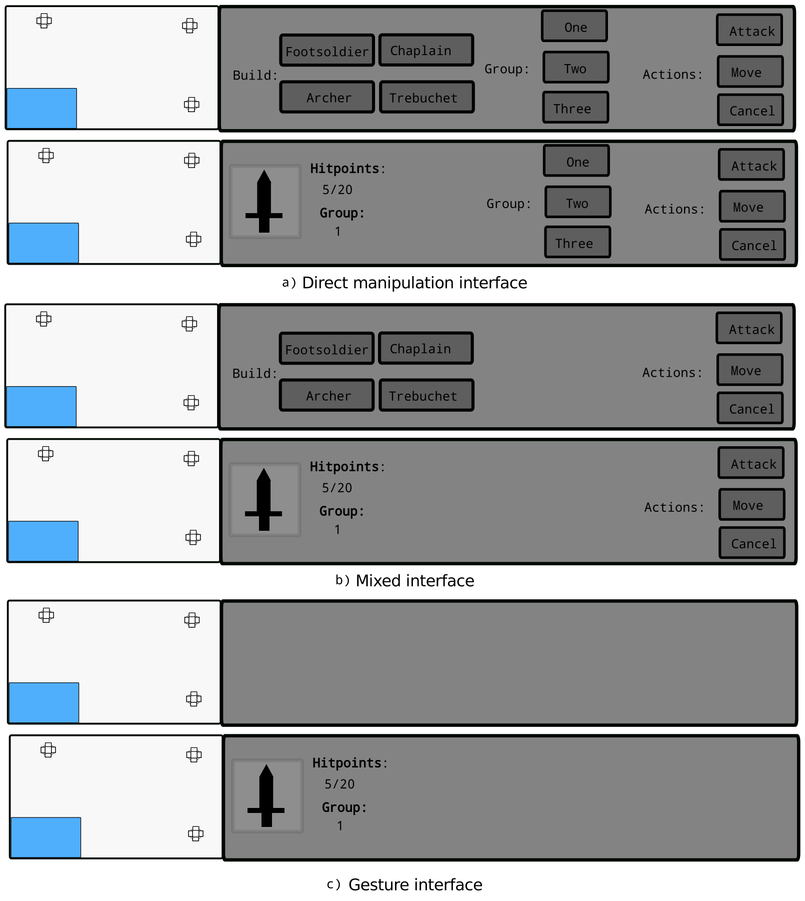


Figure 4.6: A display of the different interfaces the game will display depending on the game style. a) shows the direct-manipulation interface, where buttons allow the player to interact with unit based commands, while b) show a mixed interface, where some direct-manipulation buttons are still availabe and c) shows a gesture-based interface, where all button have been removed as the interaction are all gesture based.

4.4.3 Minimap interface

The minimap interface indicates where the player is currently with regards to the game map. The player's position is indicated by a light-blue rectangle which is displayed on top of the actual game map.

4.4.4 Direct manipulation, mixed and gesture-based interface

The games features three interface versions, one with a gesture-based interface, another with a combination of direct-manipulation and gestures and one with a direct-manipulation interface. The main differences between the direct-manipulation and gesture-based interfaces is that the direct-manipulation interface will have more user interface controls for unit behaviour than the gesture-based interface. Figure 4.8 shows a direct-manipulation interface, while figure 4.9 shows a mixed interface, combining gestures for map movement and group selecting and using buttons for unit building and behaviour commands. Figure 4.10 shows a gesture-based interface, where all buttons have been removed to allow for purely gestural interactions.

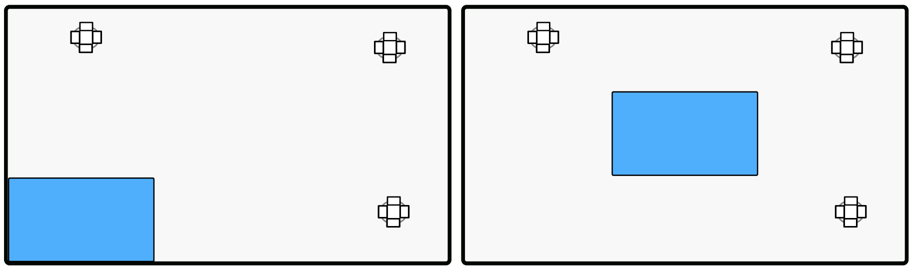


Figure 4.7: Minimap interface which is located at the bottom left of the interface

In order to indicate which gestures the player can perform on a unit or group of units, the right hand side of the screen will display the list of available gestures that the player can perform. This list is context sensitive and will change depending on the actions of the player, which allows the player to know exactly which gestures are appropriate during the course of the game. In the list of available gestures, the previously activated gesture is coloured in red, whereas the other gestures, which follow the previously selected gesture, are coloured in black, with circles around the starting points of the gesture to denote where the gesture starts and where it ends.

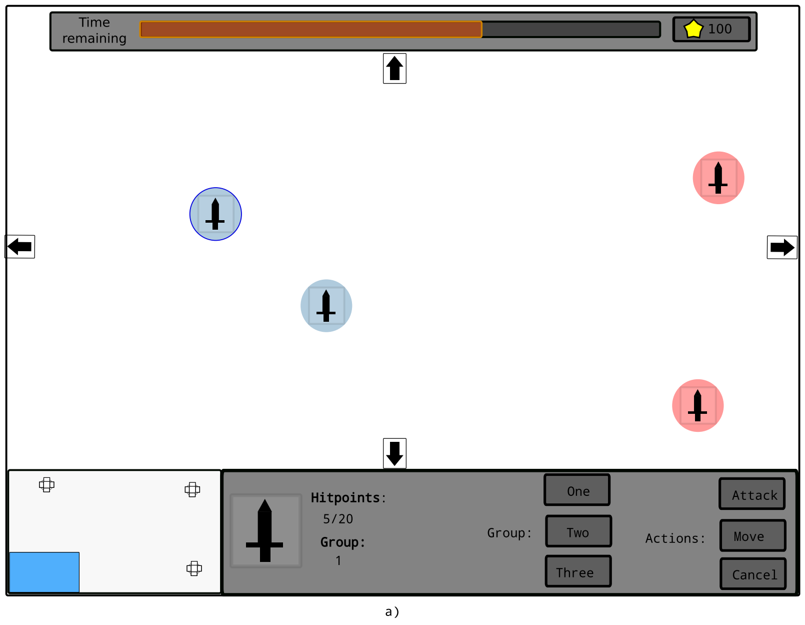


Figure 4.8: An example of a direct manipulation interface, showing a foot-soldier being selected by the player. Notice that the map movement button are displayed on the left, right, top and bottom of the game map and will move the screen in either direction depending on which button has been pressed.

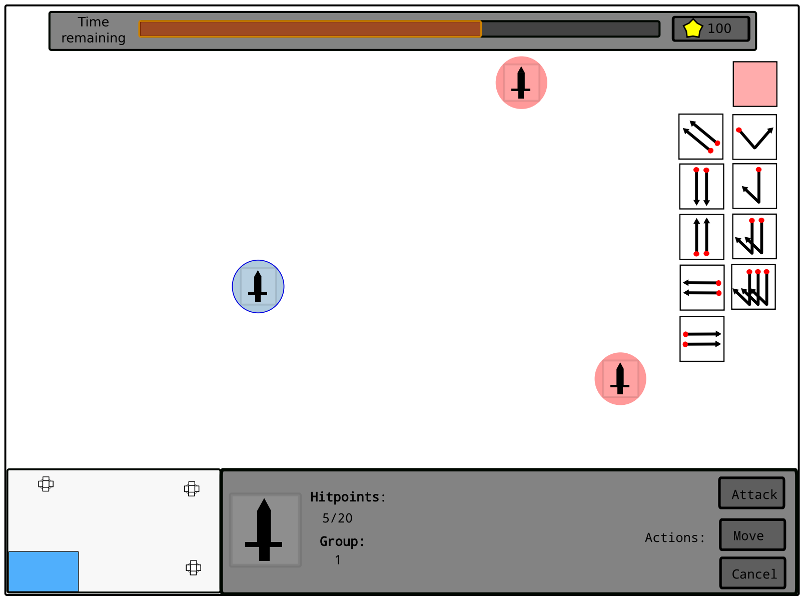


Figure 4.9: An example of a mixed interface, showing a foot-soldier being selected by the player. A list of context-sensitive gestures are displayed on the right-hand side of the screen.

4.5 Art and design

4.5.1 World design

The design of the world is shown in Figure 4.11.

4.5.2 Unit Design

The design for the units will be very minimalistic, with an emphasis on easy identification and location depending on whether the player is selecting units or finding them on the map. Figure 4.12 shows the how player and enemy units are differentiated and table 4.6 provides an elementary design of the different units available in the game.

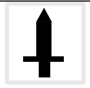
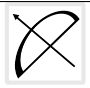
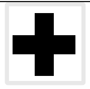
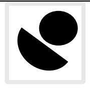
Image	Unit name
	Foot Soldier
	Archer
	Chaplain
	Trebuchet

Table 4.6: Unit style and design

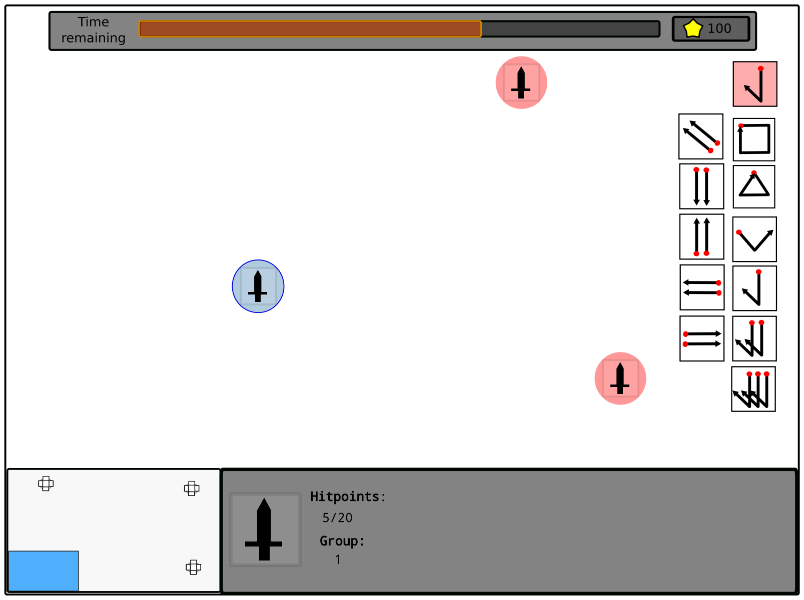


Figure 4.10: An example of a gesture-based interface, where a group one selection gesture has been performed. A list of context-sensitive gestures are displayed on the right-hand side, as well as the previously performed gesture that is show in the top right corner

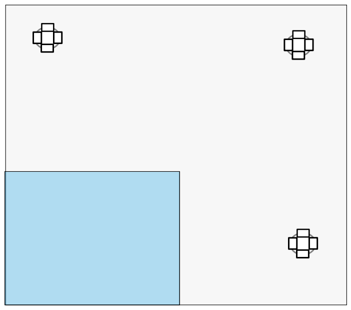


Figure 4.11: An overview of the main game environment, displaying the playing environment where game-play will occur and where the towers will be located. The blue rectangle displays the size of the display screen.

4.6 Implementation

4.6.1 Technical Specifications

Capture The Towers is designed and implemented to be played on a Apple iPad [2]. The Apple iPad has a touchscreen display with a screen resolution of 1024 x 768 pixels and supports the use of fingers instead of a styli as input mechanism. The device houses a 1 GHz Apple A4 processor, 256 MB DRAM and uses the PowerVR SGX 535 GPU for graphics.

The game was developed on Apple Mac Minis [3] running OS X 10.5 (Snow Leopard) [4]. XCode [5] was used as the main development environment and Inkscape [18] was used to produce the graphics. Five different frameworks were employed in producing the game, namely CoreGraphics, QuartzCore, OpenGL ES 1.1 [33], UIKit [1] and Foundation. Programming languages used to develop the game were C and Objective-C.

Figure 4.12: An example of how player and enemy units are displayed in the game. Player controllable units have a blue circle surrounding the unit, while enemy units are surrounded by a red circle.

4.6.2 Class Layout

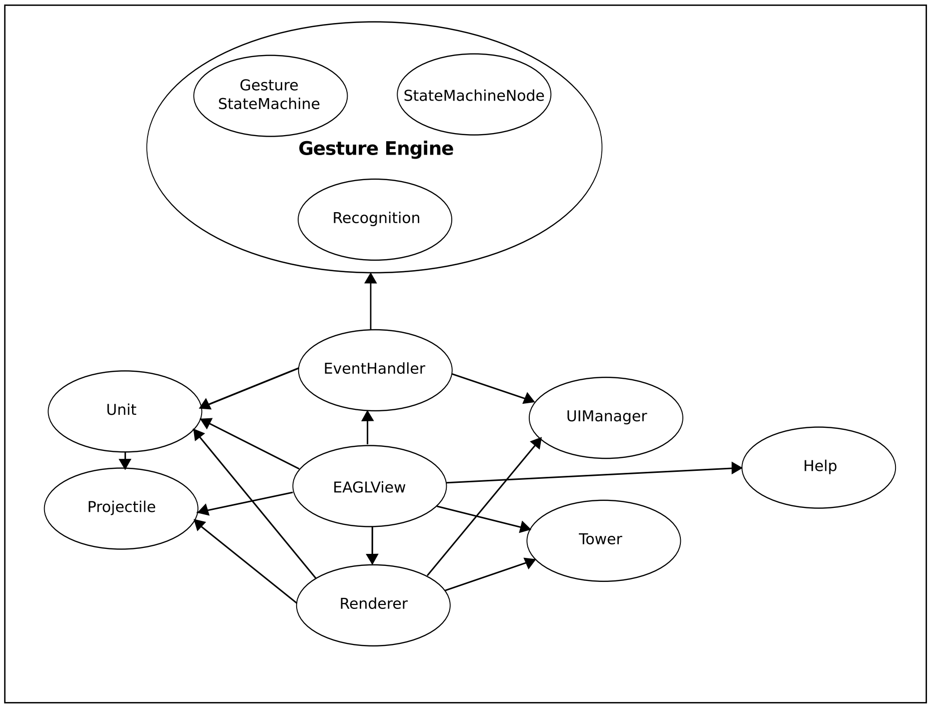


Figure 4.13: Representation of the class hierarchy used in the game. The lines between the nodes represent how each class used other classes, where the arrows indicates the direction of class use.

Figure 4.13 provides an overview of the classes used in the implementation of game. The *EAGLView* class acts as the main class and provides the game logic. The *Renderer* class is responsible for coordinating all on-screen renderings of the game world, which includes drawing of units, unit projectiles, towers and gesture interactions. The *EventHandler* class handles all events received from the player, while Unit, Projectile and Tower classes handle the specific unit, projectile and towers that are interacted with within the game. The Help and UIManager class provides general user feedback and visual feedback from the game environment.

4.6.3 EAGLView

EAGLView is a class that Apple's XCode IDE automatically sets up if a OpenGL application is created for the iPad and acts as the main ViewController for the application. With this template code, the game uses the following main loop to execute all game logic:

```
while (game_running){
    update_timers();
    event_handling();
    update_units();
    render();
}
```

Formal touch-based event-handling is done in this class, through Apple's prescribed *TouchesBegan*, *TouchesMoved* and *TouchesEnded* functions. Here, multi-touch applications detect touch specifics, such as the location touched on the screen. Basic multi-touch is supported. However, keeping track of individual touch events requires manual tracking of separate UITouch pointers. Fortunately, the game only uses a maximum of three touches. All touch events are put into a three-element NSMutable array called `touchesArray`, where each element of the array accounts for one of the three touches. The *TouchesMoved* function detects if a touch moves and adds the new touch coordinates to `touchesArray`, depending on the dimension the touch belongs to. Independent of whether a touch event is a gesture or a single touch, they all begin at *TouchesBegan* and end at *TouchesEnded*. In order to differentiate if a single touch or a multi-touch event occurred, the *TouchesEnded* functions checks the length of a particular element of `touchesArray`. If the length is less than two , then a single touch occurs, otherwise a gesture is expected. All touch events are further processed by the *EventHandler* class.

Finally, the main game-loop uses timers to ensure that basic game-play is consistent. Timers are also used to ensure that the game only lasts seven minutes and are widely used in the *Unit*, *Projectile*, *Tower* and *UIManager* classes.

Separate player and enemy unit arrays are created at the start of the game. This allows for player units to be added or removed dynamically. Accessing player or enemy units means that a iteration through the player and enemy arrays is needed, however, the number of enemy or player units in the game is small enough that it never affects performance negatively. All the enemy units and towers are created and placed at the start of the game. Similarly, three group arrays, which are used in the game to assign units to different groupings, are created at the start of the game. Enemy and player-based projectiles are also created, but a total of fifty *Projectile* objects are created within each projectile array, to ensure that no unnecessary object creation and removal is done and to ensure that the game-play is not unnecessarily slowed down. More information on how the projectiles are handled is discussed in the Projectile section.

Keeping track of world coordinates is very important for the game, not only in moving the map around, but also for displaying on-screen interactions, such as gestures drawings. The total game environment is 2050x1800, with the starting screen having its origin in the bottom, left corner. If screen movements, either by gestures or movement buttons, are performed, then the screen moves left, right, bottom and top coordinates in increments of 500. This means that the total game environment can be broken up into nine 500x500 regions, where the game screen, which has a size of (1050, 800) will overlap over the regions. The decision to do this is to show the player gradual map movement, instead of big jumps of movement, as the context switch might confuse the player. Similarly, (x, y) coordinates of touch events will need to be offset in order to interact with what is currently being displayed to the player. Figure 4.14 displays the game environment decomposition.

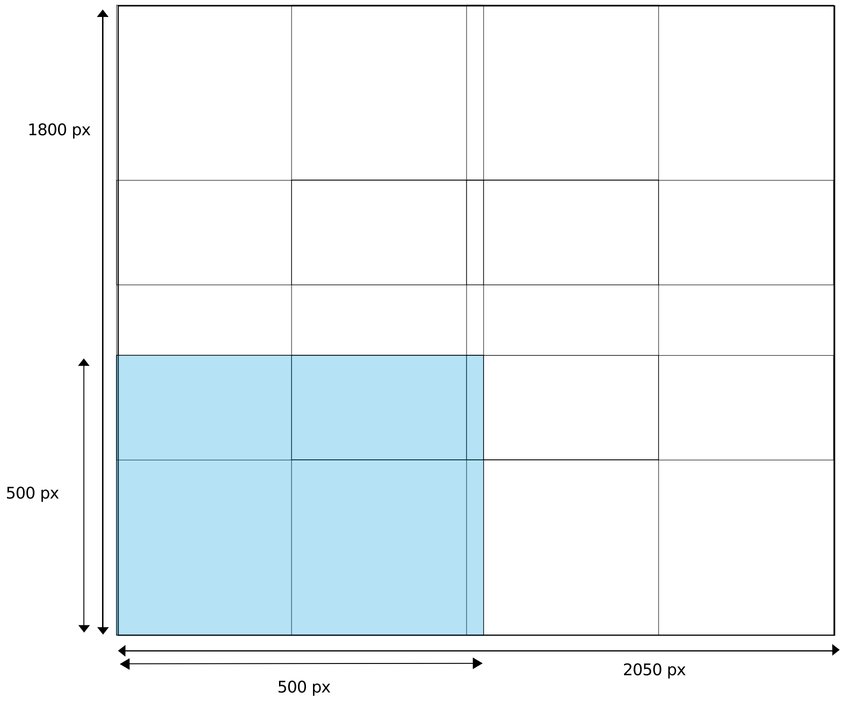


Figure 4.14: The game total game environment broken up into nine 500x500. Note that the game screen, indicated by the blue rectangle, overlaps over the nine regions.

Updating the units is done in three main steps, namely `checkInteractionCircles`, `unitCombat` and `unitStateCheck`.

The `checkInteractionCircles` function checks if units are within the interaction-circle of enemy units, or if the unit is a chaplain, if there are friendly units in its interaction-circle. However, because a unit can only attack one enemy unit at a time, a check to see if the unit is already attacking another unit is performed. If the unit is not attacking another unit, then it proceeds to check the interaction-circles. This is done by iterating through the array of units, either player or enemy, and performing a simple distance check. If the unit is a chaplain, instead of iterating through the enemy unit array, it iterates through the player array. Finally, if the unit is not moving or attacking, the distance between the unit and the towers is examined to see whether a unit is able to capture a tower.

The `unitCombat` function operates the combat mechanism between units. The function operates in a similar fashion to the `checkInteractionCircles` function, except that only attacking units are checked. Because it is already know who the target of the attacking unit is, no iterating through enemy arrays is necessary. An attack sequence is performed, which consists of generating a random number between the values of 0 and 20, and checks whether this passes the targeted unit's defence modifier. If it does, then a successful attack has happened and the targeted unit's hit-points get modified according to the attack damage it has received. That being said, further distinction is made between the foot-soldier, archer and trebuchet, as their methods of attack differ. If a foot-soldier attacks, then the normal attack sequence listed above will occur. If the unit is either an archer or a trebuchet, instead of performing a direct attack, they will launch a projectile at the position of the targeted unit. This projectile is placed in the corresponding player or enemy projectile array. After the units have been checked, then the projectiles are checked for collisions. An interation through both the player and enemy projectiles is done, which iterates through its prospective enemy unit arrays, where a distance check between the unit and projectile is done. If it is found that the radius value of the projectile and the unit is greater than the distance calculated, then an attack sequence occurs. If the projectile collides with an enemy unit, the projectile is destroy even if it does not deal damage to the unit.

The `unitStateCheck` function operates as a cleanup function and state update. It basically iterates through all the units and removes units that have 0 hit-points. Similarly, it removes references to these units from units that have them as a target, allowing units to be free to attack elsewhere. If the unit is moving, then the unit's position is updated. Similarly for the projectiles. At this stage the projectile should not have collided with a enemy unit and is thus safe to update.

The `updateStateCheck` function handles most of the user interface work. This displays the selected unit's icon in the user interface, as well as its hit points and what group it is assigned to.

UIButtons and UILabels are used to place buttons and labels on the screen. These are done by adding them as a subview to the *EAGLView* and are made visible or invisible programmatically. As UIButtons have their own event handlers, no manual touch event handling needs to be performed. OpenGL rendering is executed through the *Renderer* class.

4.6.4 EventHandler

EventHandler is an abstraction class that handles the touch events from the device and acts as a logical extension of the event-handling done in *EAGLView*. This allows the handling of touch events to be more modular and leads to a streamlined codebase.

Firstly, single touch events are handled. The main use-case for single touch events is interacting with units in the game environment. The top and bottom user interface bars do not need manual event handling, as the buttons will be handled by separate UIButtons. As such, all touches that are in the location of the top bar and the bottom bar are ignored, which means that only single touches with properties of (top - y > bottom + 150) and (top - y < bottom + 750) are handled.

If a touch event is handled, then the first check is made to see if the player is interacting with a unit on the screen, which is done by touching the unit with a finger. This will select a unit and the the user interface will change to reflect which and how many units have been selected. The process of checking whether a unit has been selected is done by performing a distance check between the position touched and the unit's centre. If a unit is interacted, then the unit will either be selected, or if it has already been selected by the player, then it will be unselected.

If a unit has not been interacted with, it means one of two interactions has occurred. Firstly, a unit command might have happened before the touch event occurred and could have taken the form of a gesture or a button press. These actions include moving a unit to a specific location or attacking an enemy unit. To differentiate between a movement command and an attacking command, a `boolean` value named `isAttacking` is passed to the *EventHandler* from the *EAGLView* class. If a move action is active, then the event handler must ensure that the single touch event does not interact with other units. Conversely, if an attack action was performed, then the single touch event must touch an enemy unit.

Lastly, a special single touch event needs to be handled: unit building. Unit building can either be done by gestures or button presses. The build type is set by using a `TYPE` variable, which is assigned depending on what unit is set to be built. Once a build type has been specified, then a unit-build area is drawn on the screen by the renderer and the player is asked to specify where they want the unit to be placed within the build area. The build area has a radius of 400, and is placed in the centre of the initial screen at (500, 500). If the distance between the touch position and the build area is less than 400, then a player unit can be placed at that location, otherwise the build area will notify the player that they have attempted to place a unit illegally.

If the game version supports gestures, then gestures are handled next. All that is needed is identifying which gesture the player has performed, seeing whether it follows the correct gesture logic within the game mechanics and performing the necessary function calls depending on the gesture. The maximum number of gestures in the game is fifteen. The only game that would use all fifteen gestures is the gesture-based interface, whereas the mixed interface only uses eleven, namely the *cancel*, *movement*, *grouping*, *attack* and *move* gestures. No extra functionality is required for gestures other than differentiating them, as all of the core functionality is also included direct manipulation interface.

4.6.5 GestureEngine

The internals of this class are as specified in Chapter 3, however, it should be noted that the *Grouped Node State Machine* engine is used. Figure 4.15 shows the *GestureStateMachine* structure used in the game, with all the gestures loaded. *GestureStateMachine* was also modified to return a string containing the gesture found.

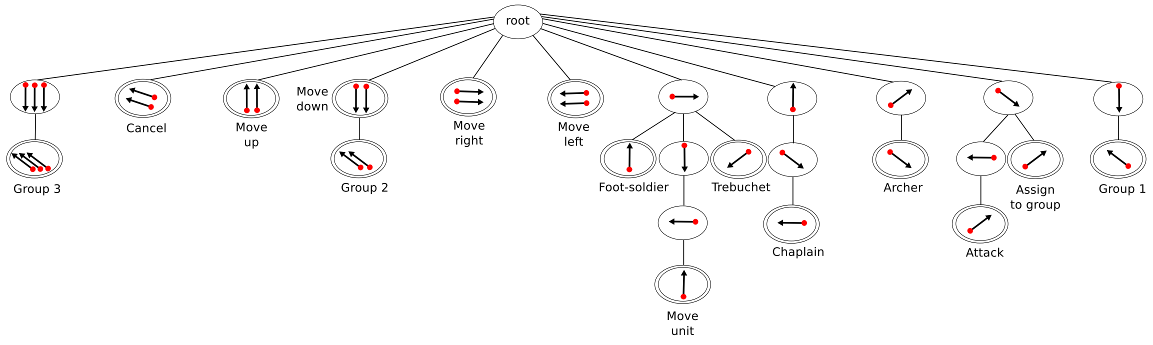


Figure 4.15: Internal representation of the *GestureStateMachine* object nodes that were used in the game.

4.6.6 Unit

The *Unit* class handles all unit functionality in the game. This includes creating units, setting the unit position in the game environment, setting the unit type, updating the unit position, receiving and sending attack damage, assigning and selecting groups, selecting individual units and rendering the unit in the game environment.

A *Unit* object functions in the game environment by use of states. This is defined by a `STATE` enumeration value as follows:

```
enum STATE{
    ALIVE,
    DEAD,
    MOVING,
    ATTACK,
    ATTACKING,
    HEALING
}
```

The default state for a unit is the `ALIVE` state. Once a unit has 0 hit-points, their state changes to `DEAD`. The `MOVING` state describes a unit that is moving, while an `ATTACK` state is activated once a unit is attacking another unit that is within its interaction-area. The `ATTACKING` state is activated once a unit has been issued an attack command from the player. The `HEALING` state only applies to the chaplain unit, which specifies whether they are healing nearby units.

Units are either friendly or enemy units. `FRIENDLY` units can only attack `ENEMY` units, and vice verse. Each unit has an `ALIGNMENT` state, defined with the following enumeration value:

```
    enum ALIGNMENT{
        FRIENDLY,
        ENEMY
    }
```

There are four different unit types available in the game (as described in section 4.3.2). The unit types are available to both `FRIENDLY` and `ENEMY` units and are defined programmatically through a `TYPE` enumeration value:

```
    enum TYPE{
        FOOTSOLDIER,
        ARCHER,
        CHAPLAIN,
        TREBUCHET
    }
```

Unit object creation initialises all position coordinates, states and timers. Unit types and alignments are also defined at object creation.

Moving a unit to a specific location is achieved by a function call from the `EAGLView` class. The input values describe the x and y coordinates the unit needs to move to. Using the unit's current position and the inputted x and y coordinates, the angle at which the unit needs to move can be calculated as follows:

$$angle = atan2(\frac{y1 - y2}{x1 - x2}) \tag{4.1}$$

Attacking an enemy unit is similar to moving a unit in the game environment, where the x and y coordinates refer to an enemy unit instead of empty space. Using equation 4.1, the angle at which the unit needs to move in order to attack a unit can be calculated.

The unit position is updated every time `updateUnit` is called. This is especially useful when a unit's state is `MOVING` or `ATTACKING`. Using the angle calculated in moving or attacking an enemy unit, the unit's x and y coordinates can then be updated by:

$$\begin{aligned} x1 &= x + movementspeed * cos(angle) \\ y1 &= y + movementspeed * sin(angle) \end{aligned} \tag{4.2}$$

Rendering a unit to the screen is done through a `render` function call from the *Renderer* class. This function draws the unit texture, selection, attack-lines and pulsing animations to the screen. The function uses a texture pointer from the *Renderer* class to bind the texture to the object being drawn.

4.6.7 Projectile

The *Projectile* class handles projectiles created by the archer and trebuchet unit types. Projectiles have a very limited lifespan which usually lasts a few seconds at most. The class handles initial projectile placements, updating of projectile positions, renderings and, in the case of the trebuchet's projectile type, a splash damage animation.

There are two types of projectiles available in the game, namely arrows or boulders. Archers create arrows and trebuchets create boulder projectiles. Projectile types are defined by a `PROJECTILE` enumeration value, which is defined as follows:

```
enum PROJECTILE{
    ARROW,
    BOULDER,
}
```

As with the *Unit* objects, projectile object functions by using lifetime states:

```
enum LIFETIME{
    ACTIVE,
    INACTIVE,
    SPLASH,
    ANIMATING
}
```

The default state of a projectile is the `ACTIVE` state. Once a projectile has collided with a enemy unit, the projectile changes to the `INACTIVE` state. Only the `BOULDER` projectile will ever be in a `SPLASH` state, and is triggered when a boulder has hit the ground and is causing splash damage. The `ANIMATING` state is for internal animation renderings of the boulder splash damage.

The initial placement of the projectile is done upon object creation. Here, the initial x and y game environment coordinates are provided, as well as the position of the target. Using the position of the target, the angle between the two points can be calculated. Equation 4.1 is then used to calculate the angle between two points, and hence, the trajectory of the projectile. This is used to update the position of the projectile in the `updateProjectile` function.

The differences between the `ARROW` and `BOULDER` projectile types is as follows. Once an arrow collides with an object, the arrow's state is set to `INACTIVE` and the arrow is then recycled internally. Once a boulder reaches a target position, instead of being switched to `INACTIVE`, the state switches to `SPLASH`. Splash damage is applied once to units that fall in a certain radius of the boulder, after which the boulder's state is changed to `ANIMATING`.

Rendering a projectile to the screen is done through a `render` function call from the *Renderer* class. This function draws the projectile and also draws the splash animation for the boulder.

4.6.8 Tower

The *Tower* class handles the tower objects required to successfully complete the game. The basic functionality that the class implements is the placement of a tower on the game environment, the timer to successfully capture the tower and the rendering of the tower.

Placing a tower in the game environment is similar to placing units and projectiles. The only difference is that the tower will never be moved from its initial position.

Timers are created once a unit is stationary over a tower. This timer is incremented every second a unit is stationary and positioned above a tower. If a unit moves, then the timer is reset and the unit will have to recapture the tower. The total time it takes the tower to be captured is 20 seconds. Once a tower has been captured, the tower changes from green to blue.

Rendering is done whenever the `render` function is called and draws the tower unit in the game environment. A `GLuint` texture pointer is sent to the `render` function from the *Renderer* class so that no unnecessary textures are loaded.

4.6.9 UIManager

The *UIManager* class is responsible for drawing the top and bottom user interface bars on the device. The top bar consists of a time-limitation bar that changes as the game progresses and the bottom bar consists of the mini-map, which gives a visual indication of where units and the viw are in the game map. Drawing is done in OpenGL and which provides a `render` method that the *Renderer* class can call.

The top bar displays the time remaining for the player to complete the game. It uses the `StartTime` and `CurrentTime` variables that are defined in *EAGLView* to display the running time in a graphical form. The bottom bar displays the current position and scale of the view in relation to the game map. Not only does it draw the entire game map with the players current location, but it also displays the towers, enemy and player units as well. The entire game map is translated to a window of width 240 and height 145. As such, all towers, units and window positions need to be translated to fit into this context.

4.6.10 Renderer

The *Renderer* class is a controller class for all the OpenGL renderings done in the game. Upon creation, the screen resolution, frame-buffers and textures of the OpenGL environment is loaded into memory. Textures are converted to the PVRTC file format as specified by Apple, which is optimised to work with the iPad's PowerVR MBX hardware. The textures are loaded into texture memory through the use of a texture pointer, and this is passed to each object that needs it.

The main rendering of the game environment occurs in the `renderer` function. This method provides the logic of which objects need to be drawn and in what order. Because the game is two-dimensional with z-axis = 0, the ordering with which each item drawn onto the screen is crucial, as OpenGL requires items at the top of to be drawn last. The sequence of OpenGL renderings is as follows:

```
renderer(){
    draw towers();
    draw player_units();
    draw enemy_units();
    draw player_projectiles();
    draw enemy_projectiles();
    draw ui_elements();
    draw multitouch_interactions();
}
```

4.7 Validation

A pilot game test was undertaken during the game development process, in which five peer Computer Science Honours students were asked to play the game and note what they found most enjoyable and most frustrating about the game, as well as providing comments and suggestions. The testing was informal and testers were asked to comment while they were playing the game. Several adjustments were made to the game to incorporate the shortcomings observed in the game testings.

One issue that was encountered early on was that of unit selection. This issue was rectified by increasing the radius of each unit. Testers also struggled to distinguish map movement gestures from other gestures, where testers would often perform a unit command incorrectly, resulting in a map movement gesture being performed, leading to confusion and frustration. This was rectified by changing map movement gestures to use two-fingers instead of one, which helped testers distinguish map movements from other unit-based actions.

Testers found the lack of visual feedback frustrating, as they were not sure if they were selecting units or operating commands correctly. This was corrected by displaying instructional messages onto the screen, such as “Assigning unit to group 1”.

Lastly, an animation to build units was incorporated into the game, which would animate screen movement to the initial map position and scale, since testers selected a build but would not know where to place the unit, as they were not in the original starting location and there was no indication where the build area was.

4.8 Conclusion

A real-time strategy game was designed and implemented on an Apple iPad and three different interfaces were developed. The game’s objective was to build units which could be directed to attack enemy units in order to capture three enemy towers. A total of four units were available for the player to build and control, each with different capabilities. Informal user testing was done and game-play adjustments were made to make the game satisfactory to the players.

Chapter 5

Testing and experiments

5.1 Introduction

The following chapter focuses on the experimental design and procedures that were undertaken in order to successfully test whether gestures are appropriate for games and if there is any differentiation of that appropriateness within game genres. The experimental specifics, such as the independent and dependant variables, how participants were selected, what the control procedures were and the process of the experimental procedure is discussed. All of the experimental designs were done by Nina Schiff [39].

5.2 Motivation

The motivation for performing user experiments is to identify whether a significance between direct manipulation, mixed and gesture-based interactions exists. With the results from a experiment like this, it can be deduced which interaction mechanism is more appropriate for games, if there are significant differences within game genres and what participant characteristics contribute to the usage of gestures within games.

5.3 Independent and dependant variables

The independent variables for the experiment is identified as *genre*, *group* and *interface type*. *Genre* refers to the game genre being played, which is either a strategy game, puzzle game or a duelling game. This independent variable is used to identify which game genre is more appropriate for each interaction style. *Group* refers to the three participant groups that are selected on the basis of a participant's appropriate technical skills and preferences, where two of the group are compared against a control group in order to validate the experiment. *Interface type* refers to the different interaction interfaces used in each game: direct manipulation, gesture and mixed interface.

The dependant variable was identified as *flow scores* and was based upon the questionnaire that the participants had to complete during the experiment. The questionnaire used was *The Flow Scales* [19] and was used to assess the state of flow that each participant was experiencing during each game for each interface. The questionnaire was obtained under license from Prof. James Gain and is mentioned in section 2.6.4.

5.4 Participants

Participants were selected from nearby Computer Labs, namely The Shuttleworth Lab and SciLab A, which are located in the Computer Science building at the University of Cape Town. Participants were presented with a form (which can be found in Appendix A). The form asked the participants to fill in their name, age, degree and year of study, as well as complete a questionnaire. The participants were also notified twenty Rand would be made available to each participant who completed the experiment. The questionnaire was in the form of a 7-point Likert scale, and questions such as computer literacy, time spent using a computer, frequency of games played, their preference of game genre and how much they enjoyed a particular game genre, if they owned a touch device and their frequency and comfortability with using a touch device, were asked. In order to keep the experiment controlled, a control group was set around computer literacy, which ensures that all participants had the same level of computer literacy. Participants that did not meet the adequate level of computer literacy were rejected from the experiment. It should be noted that adequate ethics clearance was gained by the proper authorities before any attempts were made to recruit participants for this experiment. A total of fifteen (N = 15) participants were allocated to each group member, bringing it to a total of forty five participants (N = 45), where five (N = 5) participants were allocated to each participant group.

The participants who preferred to play strategy games were chosen to play the strategy game that is presented in this report. The participants who chose strategy games as their preference had a mean value of 4.93 for how often they played strategy games (M = 4.93, SD = 1.1813), as well as a mean value of 6.26 for the enjoyment of playing strategy games (M = 6.25, SD = 0.98). The mean age of these participants was 20.133, with a mean computer literacy of 6.2, a mean gaming experience of 5.067 and a mean touch experience of 4.4. No control was made for gender or race as this did not seem to have any relevance to the experiment.

Three groups were allocated within each game genre, where five participants were allocated into each participant group. A gaming group was allocated to participants who had experience playing games. Similarly, a touch group was allocated to participants who had experience in touch devices but not necessarily in gaming experience. A control group was allocated to participants who did not have touch or gaming experience. All groups played all three game versions.

5.5 Experimental design

The experimental designed incorporated a mixed design, where repeated measures was used to pair and match participants between groups. The experiment featured 3 X 3 X 3 factorial design, where three interfaces, three genres and three participant characteristics per group was used. Lastly, the experiment used a control group to determine what effect touch and gaming experience had on the use of gestures in games.

5.6 Control procedures

Certain control procedures were followed to ensure that the experiment allowed for accurate results. Firstly, each participant had to complete a survey after each game version played. This helped counter the effects of fatigue, as the participant would have time to do another activity before playing the different game version [6], and practise effects, where the participant becomes more accomplished at the experiment the more time they spend performing the same tasks [6]. Each participant played each game version in a random sequence, meaning that the order in which they played the direct manipulation, gesture and mixed game versions was at random. This addresses the issue of learning effects, where the participant becomes more practised at the game over the duration of the experiment, and experimenter bias, where the experimenter might subject the participant towards an expected results [6]. Similarly, each participant had to write a random number to identify each game played. This number, while allowing the experimenter to identify which game version was played, served to distract the participant from identifying what the experiment was about. The participant also received written instructions on the device, minimising all interactions with the experimenter, to ensure that the participant was not biased by the experimenter and to remove any self-consciousness the participant might feel with their progress during the experiment. Demand characteristics, which refers to the demand of what the participant has to fulfil during the experiment [6], was addressed by the experimenter by not explaining the purpose of the experiment until the experiment was completed. Situational effects, where the participant responds to the experiment differently in relation to location or position, was addressed by having each participant undertake the experiment in the same setting and for the same duration. Finally, the Hawthorne effect, which relates to how a participant might modify their behaviour under the observance of the experimenter [6], was addressed by placing the participant and the experimenter in separate rooms.

5.7 Experiment procedure

The actual experiment followed a set of procedures. Before the experiment proceeded, the experimenter was responsible for contacting the participant to ensure and remind them of the time they were allocated to undertake the experiment. Once the participant arrived at the experiment location, the participant was seated at a desk where the Apple iPad and documentation, containing participant information and the questionnaire, was located. The experimenter explained the sequence of events that the participant was required to follow in order to complete the experiment, which included filling out a consent form and providing their demographic information. These forms can be found in Appendix A. The questionnaire takes the form of a *Flow State Scale* [FSS], which is used to measure flow experienced by a participant. Conventionally, the questionnaire is used to measure flow in sport and physical activity settings, but was recommended [39] to be used to measure the amount of flow participants experienced in computer games. The questionnaire consists of thirty six questions, which are divided into nine dimensions. Each dimension represents a different flow dimension. Each question asks the participant to recall a particular experience that occurred whilst playing the game, to which they respond by answering the question using a 5-point Likert response format, where 1 indicates "Strongly disagree" and 5 indicates "Strongly agree". Unfortunately, due to licensing and reproduction restrictions, the questionnaire can not be included in this report. It should be noted that the topic of flow has been discussed in this report in section 2.6.4.

Once a participant has filled out the consent and demographic forms, they continued to the device on which the games were to be played. There the participant was presented with a welcome screen, which they had to interact with in order to proceed with the next phase in the experiment. It should be noted that all instructional messages were located on the device and not on a separate piece of paper, which was meant to facilitate in the participant becoming comfortable and familiar with interacting with a touch device. All the instructional messages that were presented to the participant are located in Appendix B.

Once past the welcome screen, the participant was introduced to a practise session, where they were given the chance to practise performing gestures on the device. This was presented as a separate screen, where the participant was asked to draw the gesture presented on the screen. The gestures the participant had to perform were present in the gesture and mixed versions of the game, which meant that the participant was not being presented with different gestures in the training session and in the games, which provided a chance to learn the gestures and become comfortable in performing them before the game started. Each gesture needed to be performed ten times by the participant before proceeding to the next gesture. A total of fifteen gestures were performed by the participant before they completed the training session.

Next, the participant was presented with an introduction to the strategy game they were going to be playing. This provided a brief overview of what they could expect the game-play to be, as well as the objectives of the game and a description of the different unit types available in the game.

Once the participant had completed the description of the game, they were presented with a screen describing the game version they were about to play. This introduced the different interaction mechanisms of each game interface. They then moved on to the actual game. The game lasted seven minutes, after which the game would end with the player being presented with a cue to complete the questionnaire before proceeding. Other game versions of the game followed the same pattern. A total of three game versions were played by each participant.

Having completed all three games and questionnaires, the participant was informed that the experiment was over. The participant then left the room, handing the questionnaire to the examiner and collecting their money. An experiment explanation was presented to the participant, explaining what the experiment was about. It was found that each experiment ran for roughly forty five minutes in total, meaning no fatigue effects were likely.

5.8 Pilot experiment

A pilot experiment was conducted before the start of the main experiment process and was used to gauge the length of the experiment, as well as examine if the experimental instructions presented in the experiment were clear and intuitive. It was found that the initial time of ten minutes per game was too long and was reduced to seven minutes. Similarly, certain instructions were modified to address confusions participants of the pilot study had. A counter to indicate how many gestures the participant had completed in the practise session was also added.

5.9 Conclusion

This chapter outlines the design and procedures used to perform user experiments for this project. Independent variables, namely *genre*, *group* and *interface type*, as well as the dependant variable *flow scores* were identified and explained. Participants were selected from nearby computer labs, based on their level of computer literacy. A total of fifteen participants were allocated to each game genre, where three participant groups, namely gaming, touch and control, was allocated five participants based on their gaming and touch device experience. A 3x3x3 factorial mixed experiment design was set up, using repeated measures between groups. The control procedures for handling the experiment was describes as well has how the experiment was performed. The results of this experiment is presented and discussed in the following chapter.

Chapter 6

Results and Analysis

6.1 Introduction

This chapter describes the results drawn from the user experiments. These results include the specific results from the strategy game, as well as the results from the combination of all three games. This is done in order to investigate which interaction style best accommodates enjoyable game-play, whether the use of gestures are genre specific, and how participant characteristics influenced the the usage of gestures in game-play. All the results of the experiment, including the necessary statistical outcomes, appear in Appendix C.

6.2 Results of the strategy game

The experimental results from the flow questionnaire of the participants who played the strategy game is presented and can be found in table 10.1. Table 10.2 shows the breakdown of the flow score for each of the 9 flow dimensions. These flow dimensions, which are described in Chapter 2 of this report, are '*challenge-skill balance*', '*action-awareness merging*', '*clear goals*', '*unambiguous feedback*', '*concentration on the task at hand*', '*sense of control*', '*loss of self-consciousness*', '*transformation of time*' and '*autotelic experience*'. Table 6.1 provides a summary of the flow dimensions.

The average total flow score for the direct manipulation version of the game was 33.8 (SD = 3.93), while the gesture-based interface's average total was 31.73 (SD = 4.423) and the mixed interface was 29.68 (SD = 5.277). These results, in themselves, are not adequate to sufficiently show which version of the game outperforms the other versions and why.

To address this issue, statistical analysis, in the form of repeated measures analysis of variance [ANOVA], was performed on the data. ANOVAs were used to see if there is any difference between groups of variables and provides a statistical test for whether the means of several groups are equal. As such, it generalises t-tests to two or more groups. A repeated measures ANOVA is also used to address the issue of the same subjects being used in each treatment. The dependant variable for the ANOVA is flow, while the independent variable is game interface. Similarly, a multivariate analysis of variance [MANOVA], which is used when more than one dependant variable is present, was performed on the data to see how the interface and participant groups interacted. As well as identifying whether changes in the independent variables have significant effects on dependent variables, MANOVAs are also used to identify

Flow dimensions	Description
Challenge-skill balance	The feeling of balance between the situation and the participant's personal skill level.
Action-awareness merging	The level of involvement the participant feels, which might bring upon the sensation that their actions are automatic.
Clear goals	The feeling of certainty that the participant clearly knows what they have to accomplish.
Unambiguous feedback	The feedback the participant receives, which could lead to a feeling of confirmation as everything might be going according to plan.
Concentration on task at hand	The feeling of being really focused.
Sense of control	The distinguishing characteristic of being in complete control of the experience.
Loss of self-consciousness	The sensation where all concern for self disappears and the participant become fully part of the experience.
Transformation of time	The sensation of time either passing slowly or quickly.
Autotelic experience	The feeling of doing something for its own sake, with no expectation for reward or benefit.

Table 6.1: A summary of the different dimensions of flow

the interactions between dependent and independent variables.

The results of the ANOVA and the MANOVA applied to the results from the strategy game can be found in table 10.3 and 10.4. The results show no significance between the different interface versions, as well as no significance between the interface versions and participant groups.

6.3 Results of all three games

All three games' flow scores were combined to test for interactions between game genre, interface type and participant group. Similar to the strategy game analysis, a MANOVA was applied to the combination of games. The results of this analysis can be found in table 6.2. The results of the MANOVA showed a significance for all the main effects of the dependant variables except the combination of the interface, genre and group. In order to understand how these significant values affect each flow dimension, a univariant repeated measures test was applied for the interface on each of the nine flow dimensions, as well as post-hoc tests for the interactions between interfaces and genres and between interfaces and participant groups. A summary of the results can be found in table 6.3.

6.3.1 Interface univariate repeated measures results

The univariate repeated measures test was applied to all flow dimensions against the different interfaces. Significance was present in all nine flow dimensions. Interactions between the game interfaces and game genres were also found for the *unambiguous feedback* and *loss of self-consciousness* flow dimensions. A Tukey's range test, which is a post-hoc test performed to determine which group's means are significantly different from one another and generally incorporates a method for controlling Type 1 errors, was performed on the univariate values of

Effect	Test	Value	F	Effect df	Error df	p
Intercept	Wilks	0.009898	311.2176		28.0000	0.000000
Genre	Wilks	0.318534	2.4013	18	56.0000	0.006511
Group	Wilks	0.396239	1.8313	18	56.0000	0.043880
Genre*Group	Wilks	0.157573	1.8885	36	106.6663	0.006588
Interface	Wilks	0.099285	9.5760	18	19.0000	0.000005
Interface*Genre	Wilks	0.094389	2.3802	36	38.0000	0.004767
Interface*Group	Wilks	0.126765	1.9091	36	38.0000	0.025903
Interface*Genre*Group	Wilks	0.087785	0.9168	72	77.0695	0.644534

Table 6.2: Multivariate tests of significance on all three game. Significances are shown in red.

these results. The test results show that there were a number of significant differences between interfaces.

A comparison between the means for the *challenge-skills balance* flow dimension (table 10.6) showed that the mixed interface (M = 14.933) was greater than the direct manipulation interface (M = 11.622), which indicates that the mixed interface elicited more flow for the *challenge-skills balance* flow dimension than the direct manipulation interface. Similarly, the gesture interface (M = 15.289) had a greater mean than the direct manipulation interface.

The *action-awareness merging* flow dimension's mean comparison (table 10.8) showed that the gesture interface (M = 15.622) was greater than the direct manipulation interface (M = 13.667) and that the mixed interface (M = 15.511) had a greater mean than the direct manipulation interface, indicating that both the mixed and gesture interfaces elicited more flow for the *action-awareness merging* flow dimension than the direct manipulation interface.

The *clear goals* flow dimension mean comparison (table 10.10) showed that the mixed interface (M = 15.200) was greater than the direct manipulation interface (M = 12.578), which indicates that the mixed interface elicited more flow for this flow dimension than the direct manipulation interface. Similarly, the gesture interface (M = 15.356) had a greater mean than the direct manipulation interface.

The *unambiguous feedback* flow dimension showed an interaction between the interface and game genre dependant variables. Tukey's test (table 10.12) showed that there was significance between the different game versions and the game genres. The duelling game's direct manipulation interface (M=10.867) had a smaller mean value compared to its gesture interface (M = 15.267), meaning that its gesture interface elicited more flow for the flow dimension than its direct manipulation interface.

Control on the task at hand goals flow dimension mean comparison (table 10.14) showed that the gesture interface (M = 15.511) was greater than the direct manipulation interface (M = 13.267), which indicates that the gesture interface elicited more flow for this flow dimension than the direct manipulation interface.

Sense of control flow dimension mean comparison (table 10.16) showed that the mixed interface (M = 15.356) was greater than the direct manipulation interface (M = 12.467), which indicates that the mixed interface elicited more flow for the *sense of control* flow dimension than the direct manipulation interface. Similarly, the gesture interface (M = 15.511) had a greater mean in turn than the mixed interface, indicating that the gesture interface also elicited more flow than the mixed interface for this flow dimension.

The *loss of self-consciousness* flow dimension showed that there was an interaction between

Flow Dimension	Effect	Interface 1	>	Interface 2	Significance
Challenge-skill balance	Interface	Gestures	>	Direct manipulation	0.000111
	Interface	Mixed	>	Direct manipulation	0.000111
	Interface * genre	Puzzle gestures	>	Puzzle direct manipulation	0.000293
	Interface * genre	Puzzle mixed	>	Puzzle direct manipulation	0.004324
	Interface * genre	Duelling gestures	>	Duelling direct manipulation	0.035357
	Interface * genre	Duelling mixed	>	Duelling direct manipulation	0.023826
	Interface * group	Gaming gestures	>	Gaming direct manipulation	0.001410
	Interface * group	Duelling gesture	>	Duelling direct manipulation	0.00908
	Interface * group	Duelling mixed	>	Duelling direct manipulation	0.00405
Action-awareness merging	Interface	Gestures	>	Direct manipulation	0.001201
	Interface	Mixed	>	Direct manipulation	0.00278
Clear goals	Interface	Gestures	>	Direct manipulation	0.000114
	Interface	Mixed	>	Direct manipulation	0.000121
	Interface * genre	Strategy gestures	>	Strategy direct manipulation	0.007097
	Interface * genre	Duelling gestures	>	Duelling direct manipulation	0.001737
	Interface * genre	Duelling mixed	>	Duelling direct manipulation	0.002781
	Interface * group	Gaming gestures	>	Gaming direct manipulation	0.000855
	Interface* group	Gaming mixed	>	Gaming direct manipulation	0.004454
Unambiguous feedback	Interface * Genre	Duelling mixed	>	Duelling gestures	0.000921
Concentration on task at hand	Interface	Gestures	>	Direct manipulation	0.005088
Sense of control	Interface	Gestures	>	Direct manipulation	0.000209
	Interface	Gestures	>	Mixed	0.018505
	Interface * Group	Gaming gestures	>	Games direct manipulation	0.039190
	Interface * Group	Duelling gestures	>	Duelling direct manipulation	0.004790
Loss of self-consciousness	Interface * genre	Strategy gestures	>	Strategy mixed	0.016831
	Interface * genre	Strategy direct manipulation	>	Strategy mixed	0.0.000233
Transformation of time	Interface	Direct manipulation	>	Mixed	0.001579
	Interface * genre	Strategy direct manipulation	>	Strategy mixed	0.025650
Autotelic experience	Interface	Direct manipulation	>	Gestures	0.003083
	Interface	Direct manipulation	>	Mixed	0.00863
	Interface * genre	Strategy direct manipulation	>	Strategy mixed	0.006637
	Interface * group	Gaming direct manipulation	>	Gaming gestures	0.030455

Table 6.3: A summary of the results found for all three games.

the interface and game genre dependant variables. Tukey's test (table 10.18) revealed that there were significant differences between the different game versions and the game genres. The mean of the strategy game's mixed interface (M = 10.333) was less than its direct manipulation interface (M = 16.333), meaning the strategy game's direct manipulation interface elicited more flow for the flow dimension than the strategy game's mixed interface. Similarly, the strategy game's mixed interface's mean value was also less than its gesture interface (M = 14.6000).

The *Transformation of time* flow dimension comparisons (table 10.20) showed significant differences. The mean of the mixed interface (M = 13.156) was less than the direct manipulation interface (M = 15.556), indicating that the direct manipulation interface elicited more flow for the *transformation of time* flow dimension than the mixed interface.

Lastly, the *autotelic experience* mean comparisons (table 10.22) showed that there was a significant difference between the gesture and direct manipulation interface and between the mixed and direct manipulation interfaces. The mean of the mixed interface (M = 12.156) and the gesture interface (M = 12.444) were both less than the direct manipulation interface (M = 14.778), which indicates that the direct manipulation interface elicited more of an autotelic effect than both the mixed and gesture interfaces.

6.3.2 Interface and Genre

Tukey's range test applied to the *challenge-skills balance* flow dimension (table 10.23) showed that there were significant differences between game genres and interfaces in this flow dimension. The puzzle game's direct manipulation interface (M = 9.333) had a smaller mean value compared to the gesture (M=14.067) and mixed (M = 13.200) interfaces. This means that both the puzzle game's gesture and mixed interfaces elicited more flow for this flow dimension than its direct manipulation interface. Similarly, the duelling game's direct manipulation interface (M 12.400) had a smaller mean value compared to its gesture (M = 15.600) and mixed (M = 15.733) interfaces.

Tukey's range test applied to the *clear goals* flow dimension (table 10.25) again showed that there were significant differences. The strategy game's gesture interface (M = 14.933) had a greater mean value compared to its direct manipulation interface (M = 11.400). Similarly, the duelling game's mixed interface (M = 16.267) had a higher mean value compared to its direct manipulation interface (M = 12.467).

Tukey's range test applied to the *transformation of time* flow dimension (table 10.28) showed that there were significant differences between game genres and interfaces in this flow dimension. The strategy game's direct manipulation interface (M = 15.067) had a greater mean value compared to its mixed interface (M = 11.133), indicating that the direct manipulation interface elicited more flow for this flow dimension than the mixed interface.

Tukey's range test applied to the *autotelic experience* flow dimension (table 10.29) showed that there were significant differences. The strategy game's direct manipulation interface (M = 13.933) had a greater mean value compared to its mixed interface (M = 9.333), which indicates that the direct manipulation interface for the strategy game elicited more flow in this flow dimension than its mixed interface.

No significant results were found for Tukey's range tests applied to *concentration on task at hand* or *sense of control* flow dimensions, while *unambiguous feedback* and *loss of self-consciousness* were already examined in section 6.3.1.

6.3.3 Interface and Group

Tukey's range test applied to the interface and participant groups (table 10.30) showed that there were significant differences in the *challenge-skills balance* flow dimension. The gaming group's direct manipulation interface (M = 11.467) had a lower mean value compared to its gesture interface (M = 15.667), which indicates that the gaming group's gesture interface elicited more flow for the *challenge-skills balance flow dimension* than its direct manipulation interface. Similarly, touch's mixed interface (M = 16.667) had a greater mean value than its direct manipulation interface (M = 12.067).

Clear goals flow dimension (table 10.32) shows that there were significant differences between participant groups and interfaces in this flow dimension. The gaming group's direct manipulation interface (M = 11.533) had a smaller mean value compared to its gesture (M = 15.667) and mixed (M = 15.200) interfaces, which indicates that both the gaming's gesture and mixed interfaces elicited more flow than than its direct manipulation for this flow dimension.

Tests applied to the *sense of control* flow dimension (table 10.35) showed that there are significant differences between participant groups and interfaces in this flow dimension. The gaming's direct manipulation interface (M = 12.000) had a smaller mean value compared to its gesture interface (M = 15.133), indicating that the gaming's gesture interface elicited more flow for this flow dimension than its direct manipulation interface. Similarly, the touch group's direct manipulation interface (M = 11.800) had a smaller mean value compared to its gesture interface (M = 15.600).

Lastly, tests applied to the *autotelic experience* flow dimension (table 10.38) showed that there was a significant difference between participant groups and interfaces in this flow dimension. The gaming group's direct manipulation mean value (M = 14.667) had a bigger mean value compared to its gesture interface (M = 10.667), indicating that the gaming group's direct manipulation interface elicited more flow for this flow dimension than its gesture interface.

No significant results were found for Tukey's range tests applied to *action-awareness merging*, *unambiguous feedback*, *concentration on task at hand* or *loss of self-consciousness* flow dimensions.

6.4 Discussion

Even though the averaged flow scores for the strategy game showed that the direct manipulation version of the strategy game was preferred to the gesture and mixed game interfaces, the lack of any significance between the different interfaces for the ANOVA and the MANOVA shows that there is no clear differentiation for which interface elicits a higher flow state for the interface between the participants. These results might be due to the number of participants chosen for each game and participant groups, due to the questionnaire having nine factors and not because of the game or interface implementations.

While games considered in isolation did not produce significant differences between the interfaces, the combination of all three games did produce significance. The MANOVA for all the three game versions showed that significance for all interactions except the interaction between interface, genre and participant groups. As such, it can be said that there is no clear indication that the combination of interface, game genre and participant groups effect the flow state. However, what the significance does indicate, is that there are interactions that do influence flow. These interactions, as can be seen from the MANOVA, lie in the interactions of the interface, interface and participant groups and the interface and game genre.

The results of the univariate repeated measures, in particular the mean values for all the interactions in the Tukey's range tests, showed that the gesture interface was the dominant interface for all three games with regards to eliciting flow, followed by the mixed interface. The direct manipulation interface elicited the lowest level of flow in all three games. This means that the gesture interface provides a more compelling and enjoyable game interaction compared to the mixed and direct manipulation interfaces. The mixed interface also elicits more flow compared to the direct manipulation interface, meaning that the combination of gestures and direct manipulation elements also provide a more enjoyable game interaction compared to a direct manipulation interface. While gestures can be seen to as the preferred method of interaction for eliciting flow, caveats should be noted. For both the *transformation of time* and *autotelic experience*, the direct manipulation elicited the highest flow score. While the *transformation of time* dimension has been shown to be less important than the other dimensions [20], the *autotelic experience* is problematic, as it has been identified as being central to the flow experience [10]. One explanation for this might be due to participant familiarity with the direct manipulation interface, causing the direct manipulation interface to elicit the most flow in this flow dimension. While this is an important point to consider, it should not undermine the general preference for gestures as eliciting the most flow overall.

The comparison between interface and game genres showed that gestures are dependent on game genre. While both the puzzle and strategy games showed higher flow scores in the gestural interfaces, with the direct manipulation interface scoring the lowest, the duelling game scored the highest in the mixed interface and the lowest in the direct manipulation interface. While the strategy game scored the highest in the gestural interface, the direct manipulation interface elicited more flow in the *loss of self-consciousness*, *transformation of time* and *autotelic experience* flow dimensions. While the *transformation of time* flow dimension is of less importance compared to the other flow dimensions, the *loss of self-consciousness* and *autotelic experience* shows that the deeper flow levels were experienced in the direct manipulation interfaces. To address this issue, several considerations need to be made. Firstly, these results might be due to the implementation of the strategy game eliciting more flow in the direct manipulation interface than the other interfaces. Secondly, participants might be familiar with strategy games using direct manipulation interfaces, and as such, found that they could enter a flow state more easily with the direct manipulation interface than with the gestural interface. This would certain attribute to the *loss of self-ciousness* flow state, as participants who felt more familiar with an interface would feel less self-conscious interacting with the interface. However, these considerations fall under the realm of speculation, and as such, more experimentation is needed in order to find a clear answer. It should be noted that even though the strategy and duelling games score higher than the puzzle game with regards to overall flow score, it does not mean that the puzzle game is not more suited to gestures than the strategy game and duelling game. While the strategy game demands continuous player involvement and interaction, where units might be attacking or under attack from enemy units or the player might be moving units around the map, the puzzle provides a much slower and more strategic game-play compared to the strategy game. Similarly, the puzzle game only has the cube in continuous view, whereas the strategy game allows the player to shift focus to different areas of the map. While this is not an exhaustive list of all the differences between strategy and puzzle games, it serves to illustrate how different game genres are and how interaction styles differ. Because of the nature of the different game genres, a cross-comparison on how they interacted among different genres is not possible.

Lastly, comparisons between the interface and participant groups indicated that gestures elicited the most flow compared to the mixed and direct manipulation interfaces for the touch and gaming groups than the control group, with direct manipulation the weakest of the three interfaces. Significant differences in the gaming and touch groups indicates not only preference in the gesture interface, but also that it provided a more enjoyable interaction than the other interfaces. One caveat does appear in the gaming group for the *autotelic experience* flow state. This could be due to participants being very familiar and experienced with direct manipulation, resulting in this flow dimension to be high. The only interaction that showed significance between the two groups and the control group was found in the *transformation of time* flow dimension, where the gaming group had a larger mean value compared the the control group. Overall, the results show that the preference for gestures for both the touch and gaming groups were based on touch and/or gaming experience and not on external factors.

6.5 Summary

The results were analysed and showed that significant interactions exist for the interface, interface and genre, and interface and participant groups. The interactions on the interface showed that gestures elicited the highest level of flow among all three interfaces. However, *transformation of time* and *autotelic experience* provided the highest flow scores, indicating that the deepest flow states were elicited by the direct manipulation interface. Comparisons between the interfaces and game genres showed that both the puzzle and strategy games scored the highest flow scores with the gestural interface, with the direct manipulation interface scoring the lowest. The duelling game scored the highest with the mixed interface and the lowest with the direct manipulation interface. Compared to the other two games, the strategy game scored the highest in the *loss of self-consciousness*, *transformation of time* and *autotelic experience* flow states with the direct manipulation interface. While these flow states are indications that the strategy game elicited deeper flow states with the direct manipulation interface, these could be due to the game implementation and participant familiarity with direct manipulation interfaces.

Lastly, comparisons between the interface and participant groups indicated that gestures elicited the most flow compared to the other interfaces for the touch and gamin group than for the control group, with the direct manipulation scoring the lowest of all three interfaces. As such, it shows that the preference of gestures is dependent on touch and/or game experience and not on other factors. The only caveat appears in the *autotelic experience* flow dimension for the gaming group, where past game familiarity and experience might contribute to this flow state being high.

Chapter 7

Conclusion

This project concludes with the successful implementation of a gesture recognition engine in the form of a *Grouped Node State Machine*(chapter 3) and a game which conforms to the strategy genre. The game allows the user to control a handful of units, with the objective to capture three towers by eliminating the enemies that surround it. Three different game interfaces were developed, namely direct manipulation (where the user interacted with buttons), mixed (where a combination of gesture and direct manipulation interactions were present), and gesture-based (where the user had to perform actions by performing gestures). User experiments were conducted (chapter 5) and were controlled by computer literacy, where three participant groups were identified, namely gaming, touch and control. The gaming group included participants who were experienced with games, the touch group included participants who had touch device experience but not necessarily gaming experience, and the control group included participants who did not have extensive touch device or gaming experience. Five participants were allocated into each group, which brought a total of fifteen participants who played the strategy game. A total of three games were tested, one game per project member, with a total of forty five participants participating across the user experiments.

The results of the experiments were analysed (chapter 6), and while it was found that interaction did not exist between the combination of interface, group and genre, interactions did exist for the interface, interface and genre, and interface and participant groups. The interactions on the interface shows that gestures elicits the highest level of flow among all three interfaces. However, *transformation of time* and *autotelic experience* provided the highest flow scores with the direct manipulation interface, indicating that the deepest flow states were elicited by the direct manipulation interface. Comparisons between the interfaces and game genres showed that both the puzzle and strategy game scored the highest flow scores with the gestural interface, with the direct manipulation interface scoring the lowest. The duelling game scored the highest with the mixed interface and the lowest with the direct manipulation interface. In comparison to the other two games, the strategy game scored the highest in the *loss of self-consciousness*, *transformation of time* and *autotelic experience* flow dimensions with the direct manipulation interface. While these flow states are indications that the strategy game elicited deeper flow states with the direct manipulation interface, these could be due to the game implementation and participant familiarity with direct manipulation interfaces. Comparisons between the interface and game genres showed that the gestural interface elicited the most flow in the puzzle and strategy games, with the duelling game scoring the highest in the mixed interface. While the strategy game scored the highest in the gestural interface, the *loss of self-consciousness*, *trans-*

formation of time and *autotelic experience* flow states were highest in the direct manipulation interface, indicating that the deeper flow states were experienced with the direct manipulation interface. This could be due to limitations in the game's gestural implementation and participant familiarity with other direct manipulation strategy games. Comparisons between the interface and participant groups indicated that gestures elicited the most flow compared to the other interfaces for the touch and gaming group than for the control group, with direct manipulation scoring the lowest of all three. As such, it shows that the preference of gestures is dependent on touch and/or game experience and not on other factors. The only caveat appears in the *autotelic experience* flow dimension for the gaming group, where past game familiarity and experience might contribute to this flow state being high.

In summary, these results conclude that gestures are a more appropriate means of experiencing flow in games. As flow has been shown to be very important with respect to games and game enjoyment, it can be said that gestures provide a more appropriate means of interacting with games in order to elicit enjoyment. Gestures have been shown to be genre dependent with respect to the game genres implemented in this project and that gestures rely on the user's touch and gaming experience in order to successfully experienced.

7.1 Future Work

There are several aspects of the project that can be extended for future work.

Firstly, the gesture recognition engine can be extended. This is part feature extraction, part gesture recognition engine extension. The feature extraction could be extended to handle curves more accurately, as touch devices lend themselves to curved features as compared to a external input devices, like a mouse. By using more accurate features, the gesture recognition engine could handle more complex gestures that deal with curvature, speed, scale, dynamic multiple-touches and gesture chaining.

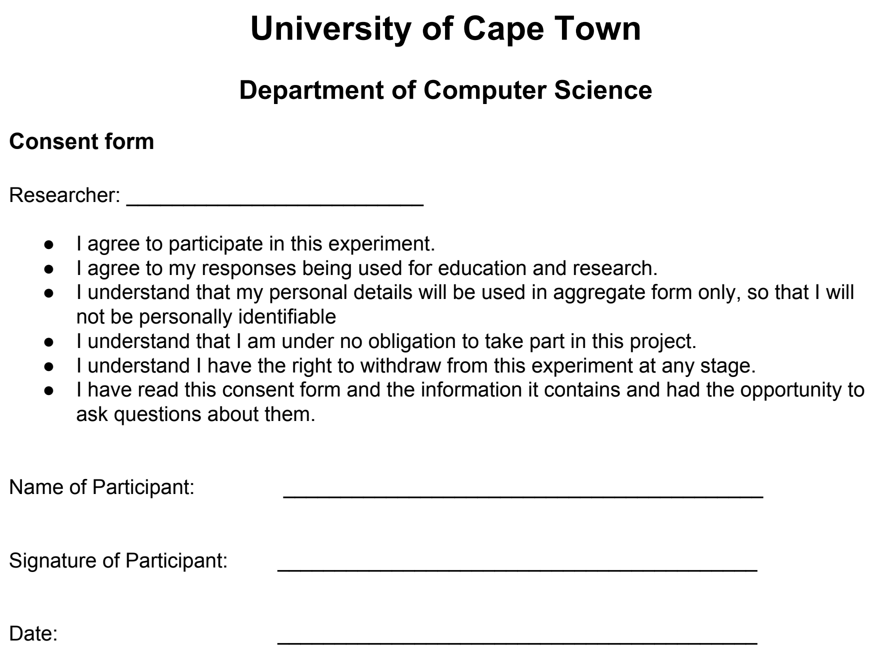
The strategy game could be extended to include more units and more sophisticated graphics. Different attack techniques could be added to each unit to give units more variety. The inclusion of a gesture recognition engine that could handle more complex gestures could be used to help identify and design gestures that are more natural and identifiable to the player and could open up larger possibilities of gestural interactions within strategy games.

Lastly, the experiment could be extended to include more participants, with longer experiment durations, in order for participants to master the games being played. The experiment could be modified to include a measure of previous game familiarity, in order to eliminate forms of interaction bias. Lastly, the experiment could be modified to asses how game genres interact with one another, as this will give a better indication of why the use of gestures are more appropriate for certain genres than others.

Chapter 8

Appendix A - Experiment forms

8.1 Consent form



University of Cape Town

Department of Computer Science

Consent form

Researcher: ____________________

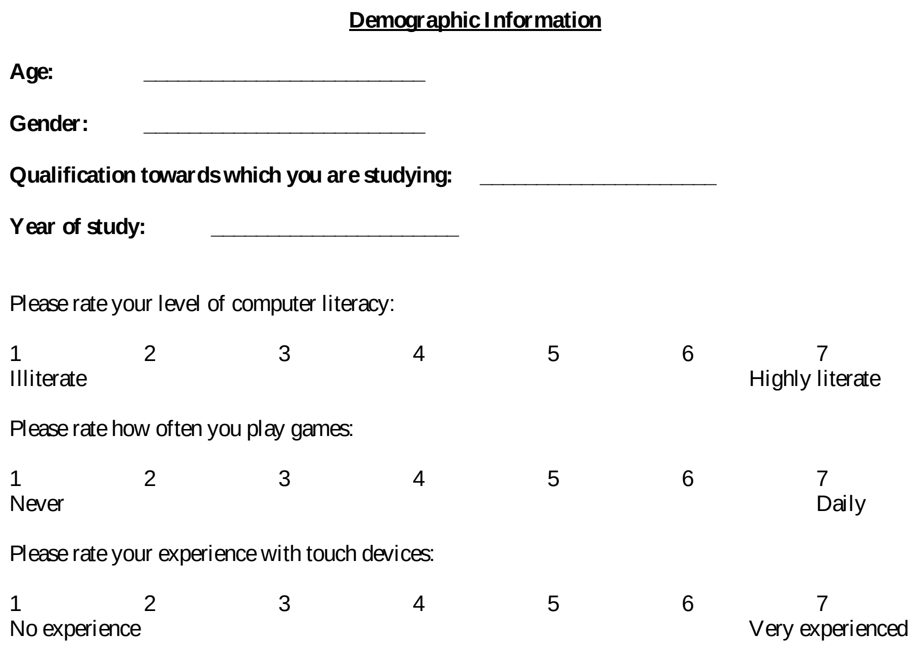
- I agree to participate in this experiment.
- I agree to my responses being used for education and research.
- I understand that my personal details will be used in aggregate form only, so that I will not be personally identifiable
- I understand that I am under no obligation to take part in this project.
- I understand I have the right to withdraw from this experiment at any stage.
- I have read this consent form and the information it contains and had the opportunity to ask questions about them.

Name of Participant: ________________________________

Signature of Participant: ________________________________

Date: ________________________________

8.2 Demographic information form



Demographic Information

Age: ______________________

Gender: ______________________

Qualification towards which you are studying: ___________________

Year of study: ____________________

Please rate your level of computer literacy:

1	2	3	4	5	6	7
Illiterate						Highly literate

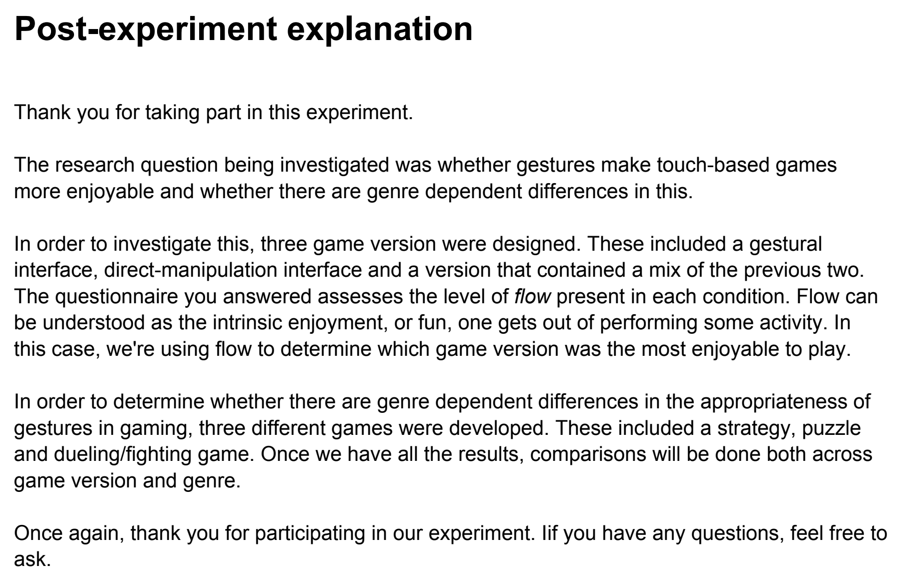
Please rate how often you play games:

1	2	3	4	5	6	7
Never						Daily

Please rate your experience with touch devices:

1	2	3	4	5	6	7
No experience						Very experienced

8.3 Post game explanation form



Post-experiment explanation

Thank you for taking part in this experiment.

The research question being investigated was whether gestures make touch-based games more enjoyable and whether there are genre dependent differences in this.

In order to investigate this, three game version were designed. These included a gestural interface, direct-manipulation interface and a version that contained a mix of the previous two. The questionnaire you answered assesses the level of *flow* present in each condition. Flow can be understood as the intrinsic enjoyment, or fun, one gets out of performing some activity. In this case, we're using flow to determine which game version was the most enjoyable to play.

In order to determine whether there are genre dependent differences in the appropriateness of gestures in gaming, three different games were developed. These included a strategy, puzzle and dueling/fighting game. Once we have all the results, comparisons will be done both across game version and genre.

Once again, thank you for participating in our experiment. Iif you have any questions, feel free to ask.

Chapter 9

Appendix B - In-game descriptions

9.1 Welcome screen

”Welcome to my iPad Game experiment. You will be required to play three game, each game lasting a total 7 minutes. Once each game has finished, it will automatically take to to the next phase of the experiment. fun and ask me any question in case you get stuck. Press the continue button to move to the next screen.”

9.2 Training session

Before we begin the actual game, let’s become familiar with some of the gestures presented in the game. The image on the left shows which gesture to draw.The red dots show the starting position and the arrows show the direction in which you draw the gesture. The red dots indicate the amount of finger you have to draw with. Draw this gesture 10 times. Take it slow and try and draw as accurate as you can.

9.3 Game description

Game Description:

The game you will be playing is a real-time strategy game. You will take control of a group of units, whose aim is to capture three enemy towers. Given a starting credit of 500, you can build a number of different units, each with their own strengths, weaknesses and abilities, which can be utilised to successfully capture the towers within the time allocated.There are four different types of units available for you to build. The foot soldier has a short attack distance, but has a strong defence and a strong attack. The archer can attack from a long distance, but has a weak defence compared to the foot-soldier. The chaplain can heal nearby units, but can not attack any enemy units. The trebuchet is a siege weapon and deals a great amount of damage, however, it is the slowest moving unit in the game and is very expensive.

The enemy towers are located in different areas on the map. Use the mini-map, which is located in the bottom left of the screen, to see where they are located. You can also move around the map by using the map movement commands. In order to capture a tower, simply place either a foot-soldier or a archer over the tower and keep them there for a 20 seconds. Once you have successfully captured a tower, you will receive 250 credit for you to build additional units.

Basic unit controls include moving units to a desired location and attack an enemy unit. Note that your units can't attack the ground, so please ensure that you select an enemy unit. Selecting units is done by performing a single tap on the unit. To unselect a unit, tap it again. You can group units into groups to allow you to move a whole bunch of units to a specific location or to attack an enemy unit.

Press the continue button once you're done to start playing the different games.

9.4 Direct manipulation introduction

Version Information:

Use the interface buttons to manipulate units and/or the screen.

9.5 After direct manipulation game

Game Over

Please write the number 627 on the front of the questionnaire.

Press continue after you have completed the questionnaire

9.6 Gesture introduction

Version Information: Gestures are used to manipulate units and/or the screen. To perform a gesture, draw the desired gesture on the screen. A help menu is available to describe the gestures available in the game. To access this help screen, tap on the credits button, which is located at the top right of the screen. A list of possible gestures are available on the right-hand side of the screen and will change depending on your gesture performed and what gestured can be performed next.

9.7 After gesture game

Game Over Please write the number 74 on the front of the questionnaire.

Press continue after you have completed the questionnaire

9.8 Mixed introduction

Version Information:

Use interface buttons and gestures to manipulate units and/or the screen. To perform a gesture, draw the desired gesture on the screen. A help menu is available to describe the gestures available in the game. To access this help screen, tap on the credits button, which is located at the top right of the screen. A list of possible gestures are available on the right-hand side of the screen and will change depending on your gesture performed and what gestured can be performed next.

9.9 After Mixed game

Game Over Please write the number 21 on the front of the questionnaire.

Press continue after you have completed the questionnaire

9.10 Experiment end

This is the end of the experiment.

Thank you so much for participating in this experiment.

Chapter 10

Appendix Control - Experiment results

10.1 Strategy game results

Participant	Direct manipulation	Gestures	Mixed
1	36.5	37.5	28.25
2	32	24	32.25
3	29	39	37.25
4	35.25	31.75	32.5
5	33.25	32	36.75
6	25.5	38.25	26
7	39.25	35	33
8	36.5	30	22.75
9	31.5	24.75	32.75
10	38.25	28.5	18.5
11	39.5	33.75	35
12	29.5	26.5	23.75
13	33.25	31	27.75
14	36.5	32	32.5
15	31.25	32	26.25
Total	33.8 (SD = 3.933)	31.73 (SD = 4.422)	29.683 (SD = 5.277)

Table 10.1: Total mean flow score between different game interfaces

Flow Dimension	Direct manipulation	Gestures	Mixed
Challenge-Skill Balance	3.367	3.167	3.03
Merging of Action and Awareness	3.3167	2.867	2.9
Clear Goals	4.183	4	3.567
Unambiguous Feedback	3.83	3.567	3.483
Concentration of the Task at hand	4.3	4.1	3.967
Sense of Control	3.4	3.05	2.83
Loss of Self-Consciousness	4.1	3.667	3.35
Transformation of Time	3.8167	3.7167	3.45
Autotelic Experience	3.483	3.6	3.1

Table 10.2: Breakdown of the flow dimensions between different game interfaces

Effect	SS	Degr. of freedom	MS	F	p
Interface	127.1028	2	63.55139	2.861109	0.076835
Interface*Group	44.5556	4	11.13889	0.501477	0.734912
Error	533.0917	24	22.21215		

Table 10.3: Univariate tests for repeated measures strategy game

Effect	SS	Degr. of freedom	MS	F	p
Group	1.29	2	0.64	0.021	0.979079
Error	362.33	12	30.36	-	-
Interface	127.10	2	63.55	2.861	0.076835
Interface*Group	44.56	4	11.14	0.501	0.734912
Error	533.09	24	22.21	-	-

Table 10.4: Repeated measures analysis of variance on the strategy game

10.2 Univariate repeated measures

Effect	SS	Degr. of freedom	MS	F	p
Interface	368.0148	2	184.0074	26.52536	0.000000
Interface*Genre	14.2963	4	3.5741	0.51522	0.724746
Interface*Group	30.5185	4	7.6296	1.09984	0.363333
Interface*Genre*Group	30.3704	8	3.7963	0.54725	0.816957
Error	499.4667	72	6.9370	-	-

Table 10.5: Univariate for repeated measure on *challenge-skill balance* flow dimension using interface, group and gesture independant variables

.

Cell No.	Interface	{1} 11.622	{2} 15.289	{3} 14.933
1	1		0.000111	0.000111
2	2	0.000111		0.798469
2	3	0.000111	0.798469	

Table 10.6: Tukey HSD test for *challenge-skill balance* flow dimension. Error: within MS = 6.9370, df=72.000. Significant values are shown in red.

Effect	SS	Degr. of freedom	MS	F	p
Interface	108.5778	2	54.28889	8.758889	0.000394
Interface*Genre	34.7111	4	8.67778	1.400060	0.242642
Interface*Group	21.6889	4	5.42222	0.874813	0.483401
Interface*Genre*Group	42.7556	8	5.34444	0.862265	0.552170
Error	446.2667	72	6.19815		

Table 10.7: Univariate for repeated measure on *action-awareness merging* flow dimension using interface, group and gesture independant variables

.

Cell No.	Interface	{1} 13.667	{2} 15.622	{3} 15.511
1	1		0.001201	0.002278
2	2	0.001201		0.975679
3	3	0.002278	0.975679	

Table 10.8: Tukey HSD test for *action-awareness merging* flow dimension. Error: within MS = 6.1981, df=72.000. Significant values are shown in red.

Effect	SS	Degr. of freedom	MS	F	p
Interface	219.2444	2	109.6222	17.49291	0.000001
Interface*Genre	50.8444	4	12.7111	2.02837	0.099458
Interface*Group	30.0889	4	7.5222	1.20035	0.318159
Interface*Genre*Group	68.6222	8	8.5778	1.36879	0.224885
Error	451.2000	72	6.2667		

Table 10.9: Univariate for repeated measure on *clear goals* flow dimension using interface, group and gesture independant variables

.

Cell No.	Interface	{1} 12.578	{2} 15.356	{3} 15.200
1	1		0.000114	0.000121
2	2	0.000114		0.953362
3	3	0.000121	0.953362	

Table 10.10: Tukey HSD test for *clear goals* flow dimension. Error: within MS = 6.2667, df=72.000. Significant values are shown in red.

Effect	SS	Degr. of freedom	MS	F	p
Interface	131.3926	2	65.69630	9.166925	0.000284
Interface*Genre	89.4519	4	22.36296	3.120413	0.020006
Interface*Group	18.7852	4	4.69630	0.655297	0.625051
Interface*Genre*Group	12.3704	8	1.54630	0.215762	0.987129
Error	516.0000	72	7.16667		

Table 10.11: Univariate for repeated measure on *unambiguous feedback* flow dimension using interface, group and gesture independant variables

.

Cell No.	Genre	Interface	{1} 13.200	{2} 13.800	{3} 15.533	{4} 14.000	{5} 16.000	{6} 15.00	{7} 10.867	{8} 15.267	{9} 12.667
1	Puzzle	1		0.999514	0.306979	0.998656	0.259906	0.809040	0.507560	0.666895	0.999936
2	Puzzle	2	0.999514		0.699209	1.000000	0.587720	0.978651	0.205774	0.930857	0.985117
3	Puzzle	3	0.306979	0.699209		0.912301	0.999977	0.999936	0.002645	1.000000	0.231775
4	Strategy	1	0.998656	1.000000	0.912301		0.517559	0.982434	0.140213	0.970262	0.959651
5	Strategy	2	0.259906	0.587720	0.999977	0.517559		0.982434	0.000663	0.999289	0.092105
6	Strategy	3	0.809040	0.978651	0.999936	0.982434	0.982434		0.012429	1.000000	0.507560
7	Duelling	1	0.507560	0.205774	0.002635	0.140213	0.000663	0.012429		0.000921	0.655002
8	Duelling	2	0.666895	0.930857	1.000000	0.970262	0.999287	1.000000	0.000921		0.181217
9	Duelling	3	0.999936	0.985117	0.231775	0.959651	0.092105	0.507560	0.655002	0.181217	

Table 10.12: Tukey HSD test for *unambiguous feedback* flow dimension. Error: between; within; Pooled MS = 9.6370, df=95.455. Significant values are shown in red.

Effect	SS	Degr. of freedom	MS	F	p
Interface	117.3778	2	58.68889	5.456612	0.006216
Interface*Genre	54.5778	4	13.64444	1.268595	0.290307
Interface*Group	24.7111	4	6.17778	0.574380	0.682091
Interface*Genre*Group	61.6000	8	7.70000	0.715909	0.676722
Error	774.4000	72	10.75556		

Table 10.13: Univariate for repeated measure on *concentration on task at hand* flow dimension using interface, group and gesture independant variables

.

Cell No.	Interface	{1} 13.267	{2} 15.511	{3} 14.022
1	1		0.005088	0.521689
2	2	0.005088		0.086483
3	3	0.521689	0.086483	

Table 10.14: Tukey HSD test for *concentration on task at hand* flow dimension. Error: within MS = 10.756, df=72.000. Significant values are shown in red.

Effect	SS	Degr. of freedom	MS	F	p
Interface	135.1259	2	67.56296	9.924918	0.000156
Interface*Genre	14.3407	4	3.58519	0.526659	0.716458
Interface*Group	63.4074	4	15.85185	2.328618	0.064197
Interface*Genre*Group	36.3259	8	4.54074	0.667029	0.718653
Error	490.1333	72	6.80741		

Table 10.15: Univariate for repeated measure on *sense of control* flow dimension using interface, group and gesture independant variables

.

Cell No.	Interface	{1} 12.467	{2} 14.889	{3} 13.356
1	1		0.000209	0.245576
2	2	0.000209		0.018505
3	3	0.245576	0.018505	

Table 10.16: Tukey HSD test for *sense of control* flow dimension. Error: within MS = 6.8074, df=72.000. Significant values are shown in red.

Effect	SS	Degr. of freedom	MS	F	p
Interface	164.0444	2	82.02222	7.719066	0.000918
Interface*Genre	182.0889	4	45.52222	4.284071	0.003637
Interface*Group	32.3556	4	8.08889	0.761241	0.553929
Interface*Genre*Group	85.1111	8	10.63889	1.001220	0.442816
Error	765.0667	72	10.62593		

Table 10.17: Univariate for repeated measure on *loss of self-consciousness* flow dimension using interface, group and gesture independant variables

.

Cell No.	Genre	Interface	{1} 13.133	{2} 15.467	{3} 13.067	{4} 16.333	{5} 14.600	{6} 10.333	{7} 13.133	{8} 13.067	{9} 12.467
1	Puzzle	1		0.575353	1.000000	0.480828	0.988243	0.657736	1.000000	1.000000	0.999963
2	Puzzle	2	0.575353		0.537515	0.999729	0.99729	0.030127	0.837733	0.815617	0.569390
3	Puzzle	3	1.000000	0.537515		0.452000	0.984363	0.686339	1.000000	1.000000	0.999983
4	Strategy	1	0.480828	0.999729	0.452000		0.871502	0.000233	0.480828	0.452000	0.230124
5	Strategy	2	0.988243	0.999729	0.984363	0.871502		0.016831	0.988243	0.988243	0.894897
6	Strategy	3	0.657736	0.030127	0.686339	0.000233	0.016831		0.657736	0.686339	0.894897
7	Duelling	1	1.000000	0.837733	1.000000	0.480828	0.988243	0.657736		1.000000	0.999753
8	Duelling	2	1.000000	0.815617	1.000000	0.4520000	0.984363	0.686339	1.000000		0.999888
9	Duelling	3	0.999963	0.569390	0.999983	0.230124	0.894897	0.999753	0.999888		

Table 10.18: Tukey HSD test for *loss of self-consciousness* flow dimension. Error: between; within; Pooled MS = 9.6370, df=95.455. Significant values are shown in red.

Effect	SS	Degr. of freedom	MS	F	p
Interface	129.7333	2	64.86667	6.616547	0.002302
Interface*Genre	44.8889	4	11.22222	1.144692	0.342549
Interface*Group	55.2889	4	13.82222	1.409898	0.239375
Interface*Genre*Group	74.8889	8	9.36111	0.954855	0.477916
Error	705.8667	72	9.80370		

Table 10.19: Univariate for repeated measure on *transformation of time* flow dimension using interface, group and gesture independant variables

.

Cell No.	Interface	{1} 15.556	{2} 14.289	{3} 13.156
1	1		0.140792	0.001579
2	2	0.140792		0.205955
3	3	0.001579	0.205955	

Table 10.20: Tukey HSD test for *transformation of time* flow dimension. Error: within MS = 9.8037, df=72.000. Significant values are shown in red.

Effect	SS	Degr. of freedom	MS	F	p
Interface	186.0593	2	93.02963	8.855279	0.000364
Interface*Genre	75.2296	4	18.80741	1.790234	0.140164
Interface*Group	42.6519	4	10.66296	1.014983	0.405488
Interface*Genre*Group	116.3259	8	14.54074	1.384100	0.218138
Error	756.4000	72	10.50556		

Table 10.21: Univariate for repeated measure on *autotelic experience* flow dimension using interface, group and gesture independant variables

.

Cell No.	Interface	{1} 14.778	{2} 12.444	{3} 12.156
1	1		0.003083	0.000863
2	2	0.003083		0.906370
3	3	0.000863	0.906370	

Table 10.22: Tukey HSD test for *autotelic experience* flow dimension. Error: within MS = 10.506, df=72.000. Significant values are shown in red.

10.3 Interface and game genre

Cell No.	Genre	Interface	{1} 9.3333	{2} 14.067	{3} 13.200	{4} 13.133	{5} 16.200	{6} 15.867	{7} 12.400	{8} 15.600	{9} 15.733
1	Puzzle	1		0.000293	0.004324	0.041253	0.000136	0.000140	0.192342	0.000152	0.000145
2	Puzzle	2	0.000293		0.992289	0.996715	0.666131	0.833858	0.885127	0.925487	0.885127
3	Puzzle	3	0.004324	0.992289		1.000000	0.216286	0.365108	0.998927	0.511983	0.436359
4	Strategy	1	0.041253	0.996715	1.000000		0.051325	0.121302	0.999449	0.473768	0.400057
5	Strategy	2	0.000136	0.666131	0.216286	0.051325		0.999994	0.041253	0.999876	0.999982
6	Strategy	3	0.000140	0.833858	0.365108	0.121302	0.999994		0.087614	1.000000	1.000000
7	Duelling	1	0.192342	0.885127	0.998927	0.999449	0.041253	0.087614		0.035257	0.023826
8	Duelling	2	0.000152	0.925487	0.511983	0.473768	0.999876	1.000000	0.035257		1.000000
9	Duelling	3	0.000145	0.885127	0.436359	0.400057	0.999982	1.000000	0.023826	1.000000	

Table 10.23: Tukey HSD test for *challenge-skill balance* flow dimension. Error: between; within; Pooled MS = 10.256, df=89.300. Significant values are shown in red.

Cell No.	Genre	Interface	{1} 14.400	{2} 14.733	{3} 16.000	{4} 12.200	{5} 15.067	{6} 14.133	{7} 14.400	{8} 17.067	{9} 16.400
1	Puzzle	1		0.999990	0.707550	0.619478	0.999710	1.000000	1.000000	0.356435	0.731314
2	Puzzle	2	0.999990		0.896785	0.427308	0.999999	0.999868	0.999999	0.541593	0.880870
3	Puzzle	3	0.707550	0.896785		0.039442	0.996519	0.798170	0.903059	0.991350	0.999994
4	Strategy	1	0.619478	0.427308	0.039442		0.056324	0.464022	0.619478	0.002246	0.014382
5	Strategy	2	0.999710	0.999999	0.996519	0.056324		0.982032	0.999710	0.731314	0.964744
6	Strategy	3	1.000000	0.999868	0.798170	0.464022	0.982032		1.000000	0.235152	0.580610
7	Duelling	1	1.000000	0.999999	0.903059	0.619478	0.999710	1.000000		0.097875	0.416919
8	Duelling	2	0.356435	0.541593	0.991350	0.002246	0.731314	0.235152	0.097875		0.998175
9	Duelling	3	0.731314	0.880870	0.999994	0.014382	0.964744	0.580610	0.416919	0.998175	

Table 10.24: Tukey HSD test for *action-awareness merging* flow dimension. Error: between; within; Pooled MS = 10.115, df=83.085. Significant values are shown in red.

Cell No.	Genre	Interface	{1} 13.867	{2} 14.733	{3} 15.667	{4} 11.400	{5} 14.933	{6} 13.667	{7} 12.467	{8} 16.400	{9} 16.267
1	Puzzle	1		0.989211	0.569355	0.501836	0.992740	1.000000	0.959756	0.464693	0.539541
2	Puzzle	2	0.989211		0.982642	0.133101	1.000000	0.992740	0.615328	0.895625	0.932726
3	Puzzle	3	0.569355	0.982642		0.016400	0.999515	0.758465	0.170510	0.999515	0.999891
4	Strategy	1	0.501836	0.133101	0.016400		0.007097	0.259318	0.992740	0.002280	0.003296
5	Strategy	2	0.992740	1.000000	0.999515	0.007097		0.899634	0.501836	0.947441	0.969875
6	Strategy	3	1.000000	0.992740	0.758465	0.259318	0.899634		0.984313	0.359334	0.428405
7	Duelling	1	0.959756	0.615328	0.170510	0.992740	0.501836	0.984313		0.001737	0.002781
8	Duelling	2	0.464693	0.895625	0.999515	0.002280	0.947441	0.359334	0.001737		1.000000
9	Duelling	3	0.539541	0.932726	0.999891	0.003296	0.969875	0.428405	0.002781	1.000000	

Table 10.25: Tukey HSD test for *clear goals* flow dimension. Error: between; within; Pooled MS = 10.667, df=80.578. Significant values are shown in red.

Cell No.	Genre	Interface	{1} 13.800	{2} 15.600	{3} 13.000	{4} 13.267	{5} 14.600	{6} 14.800	{7} 12.733	{8} 16.333	{9} 14.267
1	Puzzle	1		0.850521	0.999071	0.999982	0.999604	0.997884	0.996656	0.613771	0.999993
2	Puzzle	2	0.850521		0.435262	0.712829	0.997884	0.999604	0.444399	0.999793	0.985030
3	Puzzle	3	0.999071	0.435262		1.000000	0.954467	0.912784	1.000000	0.243183	0.989254
4	Strategy	1	0.999982	0.712829	1.000000		0.970396	0.933881	0.999982	0.350459	0.997884
5	Strategy	2	0.999604	0.997884	0.954467	0.970396		1.000000	0.894718	0.928710	1.000000
6	Strategy	3	0.997884	0.999604	0.912784	0.933881	1.000000		0.827872	0.964498	0.999982
7	Duelling	1	0.996656	0.444399	1.000000	0.999982	0.894718	0.827872		0.081952	0.933881
8	Duelling	2	0.613771	0.999793	0.243183	0.350459	0.928710	0.964498	0.081952		0.728875
9	Duelling	3	0.999993	0.985030	0.989254	0.997884	1.000000	0.999982	0.933881	0.728875	

Table 10.26: Tukey HSD test for *concentration on task at hand* flow dimension. Error: between; within; Pooled MS = 13.300, df=100.63. Significant values are shown in red.

Cell No.	Genre	Interface	{1} 13.067	{2} 15.667	{3} 13.200	{4} 12.400	{5} 14.267	{6} 13.467	{7} 11.933	{8} 14.733	{9} 13.400
1	Puzzle	1		0.156260	1.000000	0.999807	0.986972	0.999996	0.991032	0.910076	0.999999
2	Puzzle	2	0.156260		0.209600	0.178407	0.966073	0.687022	0.073159	0.997644	0.651448
3	Puzzle	3	1.000000	0.209600		0.999236	0.994032	1.000000	0.981625	0.942670	1.000000
4	Strategy	1	0.999807	0.178407	0.999236		0.576016	0.969392	0.999987	0.615103	0.996172
5	Strategy	2	0.986972	0.966073	0.994032	0.576016		0.995220	0.615103	0.999987	0.998629
6	Strategy	3	0.999996	0.687022	1.000000	0.969392	0.995220		0.942670	0.981625	1.000000
7	Duelling	1	0.991032	0.073159	0.981625	0.999987	0.615103	0.942670		0.096567	0.833036
8	Duelling	2	0.910076	0.997644	0.942670	0.615103	0.999987	0.981625	0.096567		0.894454
9	Duelling	3	0.999999	0.651448	1.000000	0.996172	0.998629	1.000000	0.833036	0.894454	

Table 10.27: Tukey HSD test for *sense of control* flow dimension. Error: between; within; Pooled MS = 11.300, df=82.059. Significant values are shown in red.

Cell No.	Genre	Interface	{1} 14.933	{2} 14.533	{3} 13.533	{4} 15.067	{5} 13.733	{6} 11.133	{7} 16.667	{8} 14.600	{9} 14.800
1	Puzzle	1		0.999993	0.948404	1.000000	0.994639	0.158720	0.945887	1.000000	1.000000
2	Puzzle	2	0.999993		0.993688	0.999987	0.999723	0.282980	0.841962	1.000000	1.000000
3	Puzzle	3	0.948404	0.993688		0.973679	1.000000	0.737052	0.392091	0.997641	0.992255
4	Strategy	1	1.000000	0.999987	0.973679		0.961020	0.025650	0.965962	0.999996	1.000000
5	Strategy	2	0.994639	0.999723	1.000000	0.961020		0.371420	0.484334	0.999501	0.997641
6	Strategy	3	0.158720	0.282980	0.737052	0.025650	0.371420		0.004822	0.258860	0.194716
7	Duelling	1	0.945887	0.841962	0.392091	0.965962	0.484334	0.004822		0.677086	0.783729
8	Duelling	2	1.000000	1.000000	0.997641	0.999996	0.999501	0.258860	0.677086		1.000000
9	Duelling	3	1.000000	1.000000	0.992255	1.000000	0.997641	0.194716	0.783729	1.000000	

Table 10.28: Tukey HSD test for *transformation of time* flow dimension. Error: between; within; Pooled MS = 14.722, df = 88.291. Significant values are shown in red.

Cell No.	Genre	Interface	{1} 15.200	{2} 12.733	{3} 13.933	{4} 13.933	{5} 12.000	{6} 9.3333	{7} 15.200	{8} 12.600	{9} 13.200
1	Puzzle	1		0.492065	0.976725	0.996510	0.528253	0.010712	1.000000	0.774791	0.937475
2	Puzzle	2	0.492065		0.983392	0.997617	0.999940	0.444092	0.820797	1.000000	0.999998
3	Puzzle	3	0.976725	0.983392		1.000000	0.948262	0.100395	0.996510	0.995031	0.999940
4	Strategy	1	0.996510	0.997617	1.000000		0.783253	0.006637	0.996510	0.995031	0.999940
5	Strategy	2	0.528253	0.999940	0.948262	0.783253		0.384045	0.528253	0.999987	0.997617
6	Strategy	3	0.010712	0.444092	0.100395	0.006637	0.384045		0.010712	0.499824	0.271660
7	Duelling	1	1.000000	0.820797	0.996510	0.996510	0.528253	0.010712		0.418945	0.750615
8	Duelling	2	0.774791	1.000000	0.995031	0.995031	0.999987	0.499824	0.418945		0.999883
9	Duelling	3	0.937475	0.999998	0.999940	0.999940	0.997617	0.271660	0.750615	0.999883	

Table 10.29: Tukey HSD test for *autotelic experience* flow dimension. Error: between; within; Pooled MS = 18.648, df = 78.186. Significant values are shown in red.

10.4 Interface and participant group

Cell No.	Genre	Interface	{1} 11.467	{2} 15.667	{3} 14.400	{4} 11.333	{5} 13.800	{6} 13.733	{7} 12.067	{8} 16.400	{9} 16.667
1	Gaming	1		0.001410	0.073441	1.000000	0.550692	0.589540	0.999876	0.001951	0.000915
2	Gaming	2	0.001410		0.923010	0.010582	0.804371	0.772556	0.065448	0.999449	0.994700
3	Gaming	3	0.073441	0.923010		0.192342	0.999876	0.999727	0.550692	0.738693	0.589540
4	Control	1	1.000000	0.010582	0.192342		0.220025	0.251707	0.999449	0.001332	0.000632
5	Control	2	0.550692	0.804371	0.999876	0.220025		1.000000	0.860821	0.400057	0.270118
6	Control	3	0.589540	0.772556	0.999727	0.251707	1.000000		0.885127	0.365108	0.242218
7	Touch	1	0.999876	0.065448	0.550692	0.999449	0.860821	0.885127		0.000908	0.000405
8	Touch	2	0.001951	0.999449	0.738693	0.001332	0.400057	0.365108	0.000908		0.999999
9	Touch	3	0.000915	0.994700	0.589540	0.000632	0.270118	0.242218	0.000405	0.999999	

Table 10.30: Tukey HSD test for *challenge-skill balanca* flow dimension. Error: between; within; Pooled MS = 10.256, df = 89.300. Significant values are shown in red.

Cell No.	Genre	Interface	{1} 13.333	{2} 15.667	{3} 16.133	{4} 13.600	{5} 15.333	{6} 14.200	{7} 14.067	{8} 15.867	{9} 16.200
1	Gaming	1		0.219195	0.068070	1.000000	0.731314	0.997937	0.999414	0.427308	0.262606
2	Gaming	2	0.219195		0.999871	0.695217	0.999999	0.939109	0.903059	1.000000	0.999946
3	Gaming	3	0.068070	0.999871		0.427308	0.998862	0.765733	0.695217	1.000000	1.000000
4	Control	1	1.000000	0.695217	0.427308		0.611501	0.999148	0.999981	0.580610	0.391191
5	Control	2	0.731314	0.999999	0.998862	0.611501		0.942941	0.974118	0.999946	0.997937
6	Control	3	0.997937	0.939109	0.765733	0.999148	0.942941		1.000000	0.880870	0.731314
7	Touch	1	0.999414	0.903059	0.695217	0.999981	0.974118	1.000000		0.562018	0.329186
8	Touch	2	0.427308	1.000000	1.000000	0.580610	0.999946	0.880870	0.562018		0.999990
9	Touch	3	0.262606	0.999946	1.000000	0.391191	0.997937	0.731314	0.329186	0.999990	

Table 10.31: Tukey HSD test for *action-awareness merging* flow dimension. Error: between; within; Pooled MS = 10.115, df = 83.085. Significant values are shown in red.

Cell No.	Genre	Interface	{1} 11.533	{2} 15.667	{3} 15.200	{4} 13.000	{5} 14.400	{6} 14.800	{7} 13.200	{8} 16.000	{9} 15.600
1	Gaming	1		0.000855	0.004454	0.947441	0.296323	0.150919	0.895625	0.009769	0.026932
2	Gaming	2	0.000855		0.999876	0.393231	0.977970	0.998312	0.501836	0.999999	1.000000
3	Gaming	3	0.004454	0.999876		0.652699	0.999057	0.999995	0.758465	0.999057	0.999995
4	Control	1	0.947441	0.393231	0.652699		0.836915	0.569355	1.000000	0.240400	0.428405
5	Control	2	0.296323	0.977970	0.999057	0.836915		0.999962	0.984313	0.915488	0.984313
6	Control	3	0.150919	0.998312	0.999995	0.569355	0.999962		0.915488	0.984313	0.999057
7	Touch	1	0.895625	0.501836	0.758465	1.000000	0.984313	0.915488		0.071049	0.194563
8	Touch	2	0.009769	0.999999	0.999057	0.240400	0.915488	0.984313	0.071049		0.999962
9	Touch	3	0.026932	1.000000	0.999995	0.428405	0.984313	0.999057	0.194563	0.999962	

Table 10.32: Tukey HSD test for *clear goals* flow dimension. Error: between; within; Pooled MS = 10.667, df = 80.578. Significant values are shown in red.

Cell No.	Genre	Interface	{1} 12.267	{2} 15.067	{3} 14.467	{4} 12.000	{5} 14.467	{6} 14.467	{7} 13.800	{8} 15.533	{9} 14.267
1	Gaming	1		0.115204	0.385606	1.000000	0.587720	0.587720	0.912301	0.106377	0.705055
2	Gaming	2	0.115204		0.999514	0.160014	0.999845	0.999845	0.970262	0.999977	0.998656
3	Gaming	3	0.385606	0.999514		0.429608	1.000000	1.000000	0.999659	0.989960	1.000000
4	Control	1	1.000000	0.160014	0.429608		0.238615	0.238615	0.809040	0.058412	0.547553
5	Control	2	0.587720	0.999845	1.000000	0.238615		1.000000	0.999659	0.989960	1.000000
6	Control	3	0.587720	0.999845	1.000000	0.238615	1.000000		0.999659	0.989960	1.000000
7	Touch	1	0.912301	0.970262	0.999659	0.809040	0.999659	0.999659		0.699209	0.999926
8	Touch	2	0.106377	0.999977	0.989960	0.058412	0.989960	0.989960	0.699209		0.929454
9	Touch	3	0.705055	0.998656	1.000000	0.547553	1.000000	1.000000	0.999926	0.929454	

Table 10.33: Tukey HSD test for *unambiguous feedback* flow dimension. Error: between; within; Pooled MS = 9.6370, df = 95.455. Significant values are shown in red.

Cell No.	Genre	Interface	{1} 13.800	{2} 16.400	{3} 13.733	{4} 13.133	{5} 15.067	{6} 14.867	{7} 12.867	{8} 15.067	{9} 13.467
1	Gaming	1		0.435262	1.000000	0.999899	0.989254	0.996656	0.998728	0.989254	1.000000
2	Gaming	2	0.435262		0.400196	0.267801	0.985030	0.964498	0.178498	0.985030	0.412069
3	Gaming	3	1.000000	0.400196		0.999955	0.985030	0.994925	0.999262	0.985030	1.000000
4	Control	1	0.999899	0.267801	0.999955		0.793870	0.875239	1.000000	0.874518	1.000000
5	Control	2	0.989254	0.985030	0.985030	0.793870		1.000000	0.773544	1.000000	0.954467
6	Control	3	0.996656	0.964498	0.994925	0.875239	1.000000		0.852215	1.000000	0.979618
7	Touch	1	0.998728	0.178498	0.999262	1.000000	0.773544	0.852215		0.657811	0.999893
8	Touch	2	0.989254	0.985030	0.985030	0.874518	1.000000	1.000000	0.657811		0.916928
9	Touch	3	1.000000	0.412069	1.000000	1.000000	0.954467	0.979618	0.999893	0.916928	

Table 10.34: Tukey HSD test for *concentration on task at hand* flow dimension. Error: between; within; Pooled MS = 13.00, df = 100.63. Significant values are shown in red.

Cell No.	Genre	Interface	{1} 12.000	{2} 15.133	{3} 13.000	{4} 13.600	{5} 13.933	{6} 14.333	{7} 11.800	{8} 15.600	{9} 12.733
1	Gaming	1		0.039190	0.979364	0.927585	0.815385	0.615103	1.000000	0.095950	0.999609
2	Gaming	2	0.039190		0.392543	0.942670	0.986972	0.999236	0.158671	0.999987	0.578317
3	Gaming	3	0.979364	0.392543		0.999913	0.997644	0.974738	0.986972	0.468555	1.000000
4	Control	1	0.927585	0.942670	0.999913		0.999993	0.997396	0.867568	0.785966	0.998629
5	Control	2	0.815385	0.986972	0.997644	0.999993		0.999972	0.721512	0.910076	0.986972
6	Control	3	0.615103	0.999236	0.974738	0.997396	0.999972		0.504734	0.981625	0.927585
7	Touch	1	1.000000	0.158671	0.986972	0.867568	0.721512	0.504734		0.004790	0.986654
8	Touch	2	0.095950	0.999987	0.468555	0.785966	0.910076	0.981625	0.004790		0.081376
9	Touch	3	0.999609	0.578317	1.000000	0.998629	0.986972	0.927585	0.986654	0.081376	

Table 10.35: Tukey HSD test for *sense of control* flow dimension. Error: between; within; Pooled MS = 11.300, df = 82.059. Significant values are shown in red.

Cell No.	Genre	Interface	{1} 15.267	{2} 14.333	{3} 12.667	{4} 12.467	{5} 14.267	{6} 10.733	{7} 14.867	{8} 14.533	{9} 12.467
1	Gaming	1		0.997037	0.426844	0.657736	0.999199	0.086526	0.999999	0.999923	0.657736
2	Gaming	2	0.997037		0.894187	0.948731	1.000000	0.318717	0.999993	1.000000	0.948731
3	Gaming	3	0.426844	0.894187		1.000000	0.979549	0.937607	0.877426	0.948731	1.000000
4	Control	1	0.657736	0.948731	1.000000		0.846225	0.871502	0.815617	0.910742	1.000000
5	Control	2	0.999199	1.000000	0.979549	0.846225		0.089924	0.999983	1.000000	0.958390
6	Control	3	0.086526	0.318717	0.937607	0.871502	0.089924		0.159815	0.250575	0.966669
7	Touch	1	0.999999	0.999993	0.877426	0.815617	0.999983	0.159815		0.999999	0.537515
8	Touch	2	0.999923	1.000000	0.948731	0.910742	1.000000	0.250575	0.999999		0.722399
9	Touch	3	0.657736	0.948731	1.000000	1.000000	0.958390	0.966669	0.537515	0.722399	

Table 10.36: Tukey HSD test for *loss of self-consciousness* flow dimension. Error: between; within; Pooled MS = 17.422, df = 82.800. Significant values are shown in red.

Cell No.	Genre	Interface	{1} 15.867	{2} 14.067	{3} 14.467	{4} 14.200	{5} 14.400	{6} 11.333	{7} 16.600	{8} 14.400	{9} 13.667
1	Gaming	1		0.815305	0.948404	0.956755	0.979996	0.042998	0.999856	0.979996	0.818120
2	Gaming	2	0.815305		0.999993	1.000000	1.000000	0.581046	0.676752	1.000000	0.999999
3	Gaming	3	0.948404	0.999993		1.000000	1.000000	0.392091	0.841962	1.000000	0.999723
4	Control	1	0.956755	1.000000	1.000000		1.000000	0.246087	0.737052	1.000000	0.999987
5	Control	2	0.979996	1.000000	1.000000	1.000000		0.172815	0.818120	1.000000	0.999856
6	Control	3	0.042998	0.581046	0.392091	0.246087	0.172815		0.008976	0.422052	0.765501
7	Touch	1	0.999856	0.676752	0.841962	0.737052	0.818120	0.008976		0.599703	0.219657
8	Touch	2	0.979996	1.000000	1.000000	1.000000	1.000000	0.422052	0.599703		0.999329
9	Touch	3	0.818120	0.999999	0.999723	0.999987	0.999856	0.765501	0.219657	0.999329	

Table 10.37: Tukey HSD test for *transformation of time* flow dimension. Error: between; within; Pooled MS = 14.722, df = 88.291. Significant values are shown in red.

Cell No.	Genre	Interface	{1} 14.667	{2} 10.667	{3} 11.467	{4} 14.267	{5} 12.000	{6} 11.267	{7} 15.400	{8} 14.667	{9} 13.733
1	Gaming	1		0.030455	0.165052	0.999999	0.750029	0.444092	0.999940	1.000000	0.999632
2	Gaming	2	0.030455		0.998988	0.365053	0.995031	0.999987	0.081621	0.231134	0.585550
3	Gaming	3	0.165052	0.998988		0.697668	0.999995	1.000000	0.250866	0.528253	0.879701
4	Control	1	0.999999	0.365053	0.697668		0.605801	0.233392	0.998434	0.999999	0.999995
5	Control	2	0.750029	0.995031	0.999995	0.605801		0.999479	0.444092	0.750029	0.972756
6	Control	3	0.444092	0.999987	1.000000	0.233392	0.999479		0.194865	0.444092	0.820797
7	Touch	1	0.999940	0.081621	0.250866	0.998434	0.444092	0.194865		0.999479	0.891110
8	Touch	2	1.000000	0.231134	0.528253	0.999999	0.750029	0.444092	0.999479		0.996918
9	Touch	3	0.999632	0.585550	0.879701	0.999995	0.972756	0.820797	0.891110	0.996918	

Table 10.38: Tukey HSD test for *autotelic experience* flow dimension. Error: between; within; Pooled MS = 18.648, df = 78.186. Significant values are shown in red.

References

[1] Apple iOS UIKit, . http://developer.apple.com/library/ios, Last Accessed: 30th Oct 2010.

[2] Apple iPad, . http://www.apple.com/ipad, Last Accessed: 30th Oct 2010.

[3] Apple Macmini, . https://www.apple.com/macmini/, Last Accessed: 30th Oct 2010.

[4] Apple OS X Snow Leopard, . http://www.apple.com/macosx, Last Accessed: 30th Oct 2010.

[5] Apple XCode IDE, . http://developer.apple.com/technologies/tools/xcode.html, Last Accessed: 30th Oct 2010.

[6] Babbie, E., and Mouton, J. *The practive of social research.* Oxford University Press, South Africa, 1998.

[7] Blake, E. Measuring presence. *CSC4000W Games and Virtual Environments course notes* (2010).

[8] Brown, E., and Cairns, P. A grounded investigation of game immersion. In *CHI '04: CHI '04 extended abstracts on Human factors in computing systems* (New York, NY, USA, 2004), ACM, pp. 1297–1300.

[9] Buxton, B. Multi-touch systems that i have known and loved (overview). http://www.billbuxton.com/multitouchOverview.html, Last accessed: 30th Oct 2010.

[10] Csikszentmihalyi, M. *Flow: the psychology of optimal experience.* Harper Perennial, New York, 1990.

[11] Dillon, R. F., Edey, J. D., and Tombaugh, J. W. Measuring the true cost of command selection: techniques and results. In *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 1990), ACM, pp. 19–26.

[12] Ermi, L., and Mayra, F. Fundamental components of the gamplay experience. *DiGRA 2005: Changing Views: Worlds in Play* (2005).

[13] Federoff, M. A., and Federoff, M. A. Heuristics and usability guidelines for the creation and evaluation of fun in video games. Tech. rep., Indiana University, Bloomington, 2002.

[14] Forlines, C., Wigdor, D., Shen, C., and Balakrishnan, R. Direct-touch vs. mouse input for tabletop displays. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 2007), ACM, pp. 647–656.

[15] Frohlich, D. M. Direct manipulation and other lessons. In *Handbook of humancomputer interaction (2nd ed* (1997), Elsevier, pp. 463–488.

[16] Hazlett, R. L. Measuring emotional valence during interactive experiences: boys at video game play. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems* (New York, NY, USA, 2006), ACM, pp. 1023–1026.

[17] Huang, T., and Pavlovic, V. Hand gesture modeling, analysis and synthesis. In *Proc 1995 IEEE International Workshop on Automatic Face and Gesture Recognition* (1995), pp. 73–79.

[18] Inkscape, . http://inkscape.org/, Last Accessed: 30th Oct 2010.

[19] Jackson, S., Eklund, R., and Martin, A. http://www.mindgarden.com/products/flow.htm, Last Accessed: 30th Oct 2010.

[20] Jackson, S., and Marsh, H. Development and validation of a scale to measure optimal experience: The flow state scale. *Journal of Sport and Exercise Psychology* (1995), 17–35.

[21] J.D, I., and Kalyanaraman, S. The effects of technological advancement and violent content in video games on players' feelings of presence, involvement, physiological arousa, and aggression. *Journal of Communication. 57*, 1 (2007), 532–555.

[22] Kang, H., Lee, C. W., and Jung, K. Recognition-based gesture spotting in video games. *Pattern Recogn. Lett. 25*, 15 (2004), 1701–1714.

[23] Kin, K., Agrawala, M., and DeRose, T. Determining the benefits of direct-touch, bimanual, and multifinger input on a multitouch workstation. In *GI '09: Proceedings of Graphics Interface 2009* (Toronto, Ont., Canada, Canada, 2009), Canadian Information Processing Society, pp. 119–124.

[24] Kratz, L., Smith, M., and Lee, F. J. Wiizards: 3d gesture recognition for game play input. In *Future Play '07: Proceedings of the 2007 conference on Future Play* (New York, NY, USA, 2007), ACM, pp. 209–212.

[25] Lessiter, J., Freeman, J., Keogh, E., and Davidoff, J. A cross-media presence questionnaire: The itc-sense of presence inventory. *Presence: Teleoper. Virtual Environ. 10*, 3 (2001), 282–297.

[26] Mandryk, R., Atkins, M. S., and Inkpen, K. A continuous and objective evaluation of emotional experience with interactive play environments. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems* (New York, NY, USA, 2006), ACM, pp. 1027–1036.

[27] McAvinney, P. Telltale gestures. *Byte*, July (2000), 237–240.

[28] McMahan, R. P., Gorton, D., Gresock, J., McConnell, W., and Bowman, D. A. Separating the effects of level of immersion and 3d interaction techniques. In *VRST '06: Proceedings of the ACM symposium on Virtual reality software and technology* (New York, NY, USA, 2006), ACM, pp. 108–111.

[29] Moscovich, T. *Principles and applications of multi-touch interaction.* PhD thesis, Providence, RI, USA, 2007. Adviser-Hughes, John F.

[30] Moscovich, T., and Hughes, J. F. Multi-finger cursor techniques. In *GI '06: Proceedings of Graphics Interface 2006* (Toronto, Ont., Canada, Canada, 2006), Canadian Information Processing Society, pp. 1–7.

[31] Moscovich, T., and Hughes, J. F. Indirect mappings of multi-touch input using one and two hands. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 2008), ACM, pp. 1275–1284.

[32] North, C., Shneiderman, B., and Plaisant, C. User controlled overviews of an image library: a case study of the visible human. In *DL '96: Proceedings of the first ACM international conference on Digital libraries* (New York, NY, USA, 1996), ACM, pp. 74–82.

[33] Open Graphics Library, . http://www.opengl.org/, Last Accessed: 30th Oct 2010.

[34] Payne, J., Keir, P., Elgoyhen, J., McLundie, M., Naef, M., Horner, M., and Anderson, P. Gameplay issues in the design of spatial 3d gestures for video games. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems* (New York, NY, USA, 2006), ACM, pp. 1217–1222.

[35] Ravaja, N., Salminen, M., Holopainen, J., Saari, T., Laarni, J., and Järvinen, A. Emotional response patterns and sense of presence during video games: potential criterion variables for game design. In *NordiCHI '04: Proceedings of the third Nordic conference on Human-computer interaction* (New York, NY, USA, 2004), ACM, pp. 339–347.

[36] Reisman, J. L., Davidson, P. L., and Han, J. Y. Generalizing multi-touch direct manipulation. In *SIGGRAPH '09: SIGGRAPH 2009: Talks* (New York, NY, USA, 2009), ACM, pp. 1–1.

[37] Rubine, D. Combining gestures and direct manipulation. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 1992), ACM, pp. 659–660.

[38] Santello, M., Flanders, M., and J.F, S. Postural hand synergies for tool use. *Journal of Neuroscience*, 18 (1998), 10105–10115.

[39] Schiff, N. An investigation of multi-touch gesture-based gaming . (honours thesis). 2010.

[40] Sears, A., and Shneiderman, B. High precision touchscreens: design strategies and comparisons with a mouse. *Int. J. Man-Mach. Stud. 34*, 4 (1991), 593–613.

[41] Shneiderman, B. Direct manipulation for comprehensible, predictable and controllable user interfaces. In *IUI '97: Proceedings of the 2nd international conference on Intelligent user interfaces* (New York, NY, USA, 1997), ACM, pp. 33–39.

[42] Sweetser, P., and Wyeth, P. Gameflow: a model for evaluating player enjoyment in games. *Comput. Entertain. 3*, 3 (2005), 3–3.

[43] Tenenbaum, G., Fogarty, G. J., and Jackson, S. A. The flow experience: a rasch analysis of jackson's flow state scale. *J Outcome Meas 3*, 3 (1999), 278–294.

[44] Watson, R. A survey of gesture recognition techniques. Tech. rep., 1993.

[45] Wigdor, D., and Morrison, G. Designing user interfaces for multi-touch and surface-gesture devices. In *CHI EA '10: Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems* (New York, NY, USA, 2010), ACM, pp. 3193–3196.

[46] Wilson, D., Dimovska, D., Selvig, S., and Jarnfelt, P. Face-off in the magic circle: getting players to look at each other, not the screen. In *ACE '09: Proceedings of the International Conference on Advances in Computer Enterntainment Technology* (New York, NY, USA, 2009), ACM, pp. 462–462.

[47] Witmer, B. G., and Singer, M. J. Measuring presence in virtual environments: A presence questionnaire. *Presence 7* (1998), 225–240.

[48] Wobbrock, J. O., Morris, M. R., and Wilson, A. D. User-defined gestures for surface computing. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems* (New York, NY, USA, 2009), ACM, pp. 1083–1092.

[49] Wood, D. Multi-touch gestures for games on the ipad. (honours thesis). 2010.

[50] Wu, M., Shen, C., Ryall, K., Forlines, C., and Balakrishnan, R. Gesture registration, relaxation, and reuse for multi-point direct-touch surfaces. In *TABLETOP '06: Proceedings of the First IEEE International Workshop on Horizontal Interactive Human-Computer Systems* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 185–192.