The Development and Implementation of an Interactive Snooker Game

Ashley Nolan

BSc (Hons) Computer Science

University of Bath

May 2006
The Development and Implementation of an Interactive Snooker Game

Submitted by Ashley Nolan

COPYRIGHT

Attention is drawn to the fact that copyright of this dissertation rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the University of Bath (see http://www.bath.ac.uk/ordinances/#intelprop).

This copy of the dissertation has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the dissertation and no information derived from it may be published without the prior written consent of the author.

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No Portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed…………………………………..

This dissertation may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purpose of consultation.

Signed…………………………….
Abstract

The computer games industry is a rapidly developing market that is consistently pushing back the boundaries of what technology can achieve. This development project explores the design and development of an interactive snooker game. The importance of current techniques utilised for this specific genre will be investigated and implemented, and any limitations placed on my own development examined. The difficulties of simulating this interactive environment are also considered, with appropriate interaction methods reviewed. The implementation and testing of the system is then presented and discussed. The dissertation concludes with an evaluation of the game produced, with logical future improvements outlined.
Acknowledgements

I would first like to thank Professor Phil Willis, whose guidance and support over the duration of this dissertation has been invaluable. I would also like to thank my friends, with special mention to Josh, Becci, Nick(s) and Luke, without whom my level of sanity would have deteriorated long ago. My continued thanks go to my family, for their unrivalled support throughout all that I do.
Contents

1. Introduction ................................................................................................................. 1

2. Literature Survey ....................................................................................................... 4
   2.1. The Games Industry ......................................................................................... 4
       2.1.1. A Brief History ...................................................................................... 4
       2.1.2. The Future of the Video Game Industry ................................................ 5
       2.1.3. Industry Trends ..................................................................................... 7
   2.2. Game Design and Usability .............................................................................. 7
       2.2.1. Game Mechanics, Dynamics, Aesthetics .............................................. 8
       2.2.2. Gameplay ............................................................................................. 9
       2.2.3. Player Interaction .................................................................................. 9
          2.2.3.1. User Interface ................................................................................. 10
       2.2.4. Player Enjoyment ................................................................................. 12
       2.2.5. Design and Usability Summary ............................................................. 12
   2.3. Graphics API ................................................................................................... 13
       2.3.1. Which API should be used for a project such as this? ........................... 13
       2.3.2. OpenGL vs. Direct3D ........................................................................... 14
       2.3.3. OpenGL – A Brief History ................................................................... 15
   2.4. Game Physics .................................................................................................. 16
       2.4.1. Why does a game need a Physics Engine? ........................................... 16
       2.4.2. Collision Detection .............................................................................. 17
       2.4.3. Collision Resolution .......................................................................... 20
   2.5. Conclusions ..................................................................................................... 21

3. Requirements Analysis and Specification .............................................................. 22
   3.1. Introduction .................................................................................................... 22
   3.2. Evaluation of Existing Products ...................................................................... 23
       3.2.1. Main Observations ............................................................................... 23
       3.2.2. Conclusions from Evaluations .............................................................. 23
   3.3. Stakeholder Requirements Analysis ............................................................... 24
   3.4. Interactivity Requirements Analysis .............................................................. 25
   3.5. MDA Framework ........................................................................................... 26
   3.6. Rule Centred Requirements Analysis ............................................................ 26
   3.7. Requirements Specification ............................................................................. 27
       3.7.1. Requirements Structure ....................................................................... 27
       3.7.2. Requirements Format .......................................................................... 28
       3.7.3. Functional Requirements .................................................................... 28
       3.7.4. Non-Functional Requirements ............................................................. 31
       3.7.5. Usability Requirements ....................................................................... 32
       3.7.6. User Requirements ............................................................................. 33
   3.8. Requirements Validation and Conclusion ...................................................... 34
## 4. Design

4.1. Prototyping .......................................................................................... 35
4.2. Design Principles .............................................................................. 36
4.3. Graphical Design ............................................................................. 36
  4.3.1. Graphically Representing the Game ........................................... 37
  4.3.2. Visual Interaction ..................................................................... 38
4.4. Game Physics .................................................................................. 40
4.5. Rule based Design ........................................................................... 41
4.6. User Interface Design ...................................................................... 42
  4.6.1. User Interface Design Guidelines ............................................. 42
  4.6.2. Output considerations .............................................................. 42
  4.6.3. Input considerations ................................................................. 46
4.7. Design Conclusion .......................................................................... 47

## 5. Implementation

5.1. Implementation Language and API ................................................... 49
5.2. Pre-Implementation Work ............................................................... 50
5.3. Code Architecture .......................................................................... 50
5.4. System Implementation .................................................................. 51
  5.4.1. Visual Implementation .............................................................. 51
  5.4.2. Physics Implementation ........................................................... 54
  5.4.3. Rule Implementation ................................................................. 59
  5.4.4. User Interface Implementation .................................................. 62
5.5. Implementation Conclusion ............................................................. 64

## 6. System Testing and Evaluations

6.1. Test Plan .......................................................................................... 65
6.2. Defect Testing .................................................................................. 66
  6.2.1. Black-Box Testing ................................................................... 66
  6.2.2. White-Box Testing .................................................................. 69
6.3. Integration Testing ........................................................................... 70
6.4. User Evaluations ............................................................................. 71

## 7. Conclusions

7.1. Future Improvements .................................................................... 75
7.2. Concluding Remark ........................................................................ 76

## 8. Bibliography

8.1. References ....................................................................................... 77

### Appendices

A. Evaluation of Existing Snooker Games ............................................. 80
  A.1. World Snooker Championship 2005 (WSC) .................................. 80
  A.2. FooBilliard .................................................................................. 82
B. Stakeholder Analysis Questionnaires ............................................... 85
C. The Rules of Snooker ....................................................................... 87
  C.1. The Standard Table ..................................................................... 87
  C.2. Definitions .................................................................................. 88
  C.3. The Game ................................................................................... 89
D. Usability Principles and Guidelines .................................................. 95
  D.1. Design Principles ........................................................................ 95
  D.2. Graphic Design Principles .......................................................... 96
E. System Screenshots of Developmental Stages ................................... 97
F. Testing and Evaluation ..................................................................... 100
  F.1. Defect Testing ............................................................................. 100
  F.2. Integration Testing ...................................................................... 108
  F.3. Evaluations .................................................................................. 109
G. Program Listings ............................................................................... 112
Chapter 1

Introduction

Interactive entertainment has grown at a tremendous rate over the last fifteen years. The computer games industry in particular has thrived in this period, becoming a multi-billion-pound industry from what was once a niche market. Video game development continues to push the latest technology to the limit, driving the advancement of high-end graphics and Artificial Intelligence.

Video games are renowned as one of the more creative forms of software development. The industries huge growth in audience over the last fifteen years can be at least partly attributed to its origins in promoting creative and imaginative development. Highly skilled developers are drawn to the prospect of creating a virtual world that potentially thousands of people can experience and lose themselves in. This is just one of the factors that make the industry so captivating and was the initial inspiration for the conception of this project.

So what challenges are faced when undertaking a project in this domain, and what are the key considerations when creating a virtual world with which users can interact? These two questions will drive the development of this project and we hope to have gained a better understanding of the process of game development upon its successful completion.

Snooker has been chosen as the focus of the game to be developed as it provides a platform from which we can investigate a key problem domain within game development; the translation of a real world activity into an interactive game world. The problems relating to this range from how real world entities can be visually represented in a game, right through to the interaction experience itself, identifying the key aspects of the activity that must be conveyed in its virtual representation. These issues will be confronted and sensible design solutions investigated.

It is important to realise that the limitations on the development of this project are far different to those faced in a commercially produced project. Games projects can command vast budgets, with development usually stretching over several years carried out by teams consisting of 20-50 individuals. This project will be carried out by a sole developer over roughly a seven month
timescale. Consequently, it is unrealistic to expect the game produced to be of a commercial standard. Therefore the goal of the project will be to produce a game that realistically represents the game of snooker, providing an experience that users can relate to the real life activity. The specific features the game will encompass will be determined by exploring the key goals of the game and identifying possible design and implementation issues, considering what can be feasibly achieved within the projects timescale.

1.1 Aims

- Design and implement an interactive snooker game, implementing the basic functionality needed in a game of this kind.
- The game should provide a realistic experience to users, accurately mirroring the key elements present in a real-life game of snooker. Examples of these include the game visuals, game physics and user interaction with the game.

1.2 Objectives

- Research the problem domain, focusing on the key aspects relating to the design and development of an interactive computer game. This includes investigating the potential solutions for providing graphics and physics in the game.
- Research and evaluate currently available games in this genre to obtain an understanding of the approaches utilised in translating snooker to a virtual medium.
- Analyse the opinions of potential system stakeholders, gathering and prioritising the requirements of the game.
- Design and implement an interactive snooker game that encompasses the features required to present a realistic interaction experience.
- Test, throughout development and upon completion, the functionality of the game, resolving erroneous test cases.
- Analyse how people use the game, in particular studying their interaction. Revise and develop interaction elements of the system, such as the control system, to ensure the game achieves a good standard of accessibility.
- Draw clear conclusions to the successes and failures of the project, suggesting logical areas of improvement.

1.3 Lifecycle Methodology

This project follows an adapted Incremental development approach (1). In essence, users will identify the services to be provided by the system, identifying the relative priority of each service. A number of increments are then defined, each providing a subset of system functionality. The services are integrated into each increment in respect to its pre-defined priority. This methodology enables the system functionality of most importance to be implemented first, but the requirements and design solutions of lower priority features are also identified. This ensures low priority features can be
implemented in future increments without needing to go back through the requirements and design development stages.

In the game being produced for this project, these increments will be relative to the features identified as the highest priority through requirements gathering. Once these components have been implemented, lower priority functionality will then be incorporated into the game. The number of increments fulfilled will depend on the speed of the development process, with as many increments being integrated into the system as the project constraints will allow time for. Users will be asked for feedback after each system increment to allow the user to clarify how future increments should be integrated with the current implementation, as well as identifying aspects of improvement within the current system implementation.
Chapter 2

Literature Survey

Before being able to consider the requirements for the development of an interactive game, it is important to explore the potential challenges that will be faced developing a project of this nature. This Literature Survey will first investigate how games have developed over recent years, in particular identifying the trends of the computer game industry during this time. It will then examine some key aspects of game design, including the specific challenges of designing for interactive games. Finally, the technical challenges of the project shall be considered, in particular the game physics.

2.1. The Games Industry

2.1.1. A Brief History

The first video game ever made is credited to William Higinbotham, a member of Brookhaven National Laboratory, in 1958. Running on an analog computer and hooked up to an oscilloscope, it was a side view tennis game called Tennis for Two. Although several hundred people saw the Brookhaven exhibit in 1958/59, it did very little to inspire future video games. This can be mainly attributed to it failing to reach the right audience to make an impact. Tennis for Two was dismantled shortly after the autumn of 1959, and replaced with newer exhibits the following year, as Higinbotham didn’t consider his invention a major breakthrough (2).

Tennis for Two was followed by a few other relatively low-key productions, such as Steve Russell’s Spacewar, an action space game created in 1961 (2). This was soon to change due to Nolan Bushnell, an engineer who had come into contact with Spacewar during his education at the University of Utah. In 1969, Nolan failed in his attempt to launch his own adaptation of Russell’s Spacewar, Computer Space, coming to the conclusion that the game was too complex to play. Bushnell therefore set about trying to come up with a simpler idea that could have mass appeal. In 1972, wanting a bigger cut of whatever he designed, Bushnell set out to start his own company to design video games.
This company was called Atari. Several months later, Atari produced what would become the first renowned video game, *Pong*. This simple 2D tennis game is seen as effectively launching the video game industry on a global scale.

From 1977-1982, the video game evolution continued with the development of the cartridge based console and several arcade classics, such as *Space Invaders, Donkey Kong* and *Pac-man* (which went on to become the biggest arcade game of all time) in what was heralded ‘The Golden Era’. However, the industry did not continue on this path of success. In 1983, the video game market crashed, mainly due to the influx of cheap home computers such as the Commodore 64 (2). These not only duplicated many of the popular console games but also offered software such as word processing and accounting programs. This competition, coupled with a bad economy and simply too many products being made available, resulted in the crash. Products were dumped cheaply as consumers began to lose confidence in the industry, with many companies leaving the video game business altogether.

The first company to kick-start the next phase of the video game industry was Nintendo when their first console, the Nintendo Entertainment System (NES), was released in New York, in 1985 (3). Although retailers were reluctant at first to stock a home video game system, Nintendo positioned the console as an “entertainment system” as well as guaranteeing retailers that they would buy back any unsold systems. Once on the shelves, the NES became a hit in a limited market, helped by the success of original games such as Super Mario Bros, Donkey Kong, Metroid and The Legend of Zelda. The NES was released nationwide in 1986, and its success reached such heights that at times in the late 1980’s, Nintendo held over 90% of the video game market (2).

Nintendo soon found a rival in the form of Japanese company Sega. Sega’s first foray into the console market was the Sega Master System in 1984, although it was their second console, the Sega Mega Drive (released in late 1990) that proved the greater success. Nintendo released their second console, the Super NES in 1991 and the console evolution had truly begun.

Over the next 12 years, consoles evolved with remarkable speed, with new companies coming to the fore and older competitors being forced out of the market. Sega released their last home console, the Dreamcast, in 1999, but by then the Sony PlayStation (1995) and Nintendo’s N64 (1996) had strong footholds in the market and the Dreamcast never really found its niche. Sony have gone on to become the market leader in the video game industry, with the Playstation 2 (2000) dominating the market over its competitors, the Nintendo Gamecube and Microsoft’s Xbox.

2.1.2. The Future of the Video Game Industry

Personal Computers have become much more adept at handling video games due to the evolution of console gaming. This has shown hardware vendors that there is a growing market for more powerful hardware to run increasingly more demanding games. The PC game market, while not as big as the console game market, holds a sizeable market share with some of the most popular game franchises available at present hailing from the PC game market. Such games
include Half-life, Doom, World of Warcraft and Football Manager, all of which have outsold many popular console franchises over the last year.

The industry is currently entering the next generation of console gaming, with the Microsoft 360 released last December, and the Playstation 3 and Nintendo Wii to be released before the end of the year. With each new generation of consoles, the possibilities available to developers grow, and the scope of each project gets far larger.

As consoles evolve, one of the most noticeable areas of advancement is the visual aspect of video games. Over the past 20 years the industry has seen key innovations in computer graphics, with texture mapping, 3D modelling, and ever rising polygon counts. Arguably the biggest innovation was the switch from 2D to 3D in game development, this coming as recently as 1995 for games consoles, with the release of the Playstation (3).

Graphics have developed to astounding levels of realism and quality with the improvement of video hardware. Many believe this improvement has happened at such a breakneck pace over the last 30 years that graphical improvement will become less important to what drives the type of games that people make (4). This is mainly because, when visually compared, the room for graphical improvement will become narrower and thus less important to actually improving the quality of the games being developed. After all, once photo realism is achieved, what level of realism can developers aim for?

This opinion was recently supported at the Game Developers Conference 2003, in San Jose, California, by Jason Rubin. Rubin is the founder of developer Naughty Dog, a games company well regarded for pushing systems to their graphical limits. Rubin presented his lecture (4) on the fact that graphical development in the industry, while still important, would become less of a driving force behind what types of games developers make and subsequently sell. One article on this lecture (5) goes on to try and answer Rubin’s final question posed to his fellow developers; “What are we going to do to expand what we do?”  The article suggests that Artistry - the way game story is written and presented - and Interactivity - the way in which someone interacts with the game - are perhaps the way forward.

Interactivity is perhaps the more interesting of these two suggestions, as it is this that will set games apart when developing games for the next generation of consoles. Developers are now focusing their efforts more towards innovation in the way games are played rather than the way they are seen. This is exemplified by Nintendo with their current and future games systems, the Dual-Screen and the Wii (6). With these two systems Nintendo have shown that they are looking to change the way games are played, rather than focusing on the previously successful wow-factor of graphical improvement.

From a slightly different perspective on interactivity, Xbox 360 is concentrating on the interactivity via network play with their primary objective to make online gaming much more accessible. Users can play games with or against other people in specially developed multiplayer modes. Leagues can also be put in place for certain games to make playing more competitive and find the best game players across the world.
Whatever happens in the future, the video game industry is currently riding a wave that has no apparent end in sight. This makes it an exciting industry to be involved in, with technological and developmental enhancements being realised with every new game release. Price Waterhouse Coopers predicts that the game industry will globally grow at a 20 percent annual growth rate through 2008, raising the industries 2003 revenues of $22.3 billion to a projected $55.6 billion (2). This far exceeds the projected growth rate of any other media industry.

2.1.3. Industry Trends

Looking over the history of video games, many things have changed. However a number of considerations still remain:

**Graphics** have always been the most noticeable advancement in video games. Some games now come close to photo-realism, which is a huge progression from the visuals games, such as Pong, exhibited just 35 years ago. In terms of this project, although photo-realism is not a realistic goal, the graphics must clearly represent a snooker table, using the most appropriate tools available to give a realistic impression of the game. Section 2.4 discusses what tools could be utilised in the development of a project of this nature.

**Interactivity** has become something of a buzzword over the last decade in games. It looks set with the new generation of consoles to take over as the key differentiator in what consumers want to see from games. Interactivity is especially important when developing a game based on real life activities, as users know how the game world should behave. Interactivity is discussed in more detail in section 2.2.3.

**Usability** has always been a key consideration in the development of a game. If a game is too difficult to use, people will not use it irrespective of how good the other aspects of the game may be. Nolan Bushnell first realised this when releasing *Computer Space* in 1972, and this still applies in the games being developed today. Usability must be achieved on a number of different levels and is investigated in more detail below.

2.2. Game Design and Usability

Before starting any kind of development, it is a worthwhile practice to first research the aspects of design that will be considered most crucial to the project. This section therefore investigates some of the key game design considerations, relating them more specifically to issues that should be considered when designing for this project in particular.

One of the key considerations of any development is usability. Dictionary.com (7) defines usability as ‘The effectiveness, efficiency and satisfaction with which users can achieve tasks in a particular environment of a product. High usability means a system is: easy to learn and remember, efficient, visually pleasing and fun to use and quick to recover from errors’. It is apparent that usability should be an attribute that every system attempts to achieve, but in reality very few systems accomplish usability on every level.
The usability of this project will need to be tackled carefully. Much of the literature on usability in video games presents many heuristics for designing and evaluating games, but this section will focus on three key aspects. These are Game Mechanics (interaction with the game world), Gameplay (problems and challenges) and Interactivity (User Interface: front-end and controls). We will first look at these key design concepts, before briefly looking at how we can attempt to measure player enjoyment to help create more enjoyable games.

2.2.1. Game Mechanics, Dynamics, Aesthetics

‘Game mechanics’ is a very broad term for a much larger spectrum in game design. Wikipedia (8) defines a game mechanic as ‘a rule or set of rules intended to produce a set of outcomes in a game…The interaction of the various game mechanics in a game determines the complexity and level of player interaction in the game’. A set of game mechanics are what makes up the gameplay of a game, and are therefore key to defining how good or bad an experience the user has when playing the game. Ultimately, if the player has no way of achieving their goals in the game, why would they continue to play?

There are a number of articles relating to what produces a worthwhile gaming experience. Many of these look towards utilising the MDA Framework (36); standing for Mechanics, Dynamics and Aesthetics. These are defined in relation to this framework as follows:

- **Mechanics** are the various actions, behaviors and control mechanisms offered to the player within the game context.
- **Dynamics** describes the run-time behavior of the mechanics acting on player inputs and each others’ outputs over time.
- **Aesthetics** describes the desirable emotional responses evoked by the game dynamics.

MDA supports a formal iterative approach to design and tuning, allowing the explicit consideration of particular design goals, foreseeing the impact of changes on each part of the framework and the consequent designs and implementations. It is helpful to consider the perspectives of the MDA framework from the perspective of the designer and player (9) (see figure 2.1.). For the designer, the mechanics bring about dynamic behavior, which in turn leads to particular aesthetic experiences. Alternatively, from a player perspective, aesthetics set the tone, which are represented by the visual dynamics and operable mechanics.

![Fig 2.1. Perspectives on the MDA framework (9)](image)

In a presentation shown at the International Game Developers Conference 2004, Marc LeBlanc (10), the conceptual creator of the MDA framework, reiterated this point, stating that designers should view this model by looking at what provides good aesthetics, then working backwards to achieve the
appropriate game dynamics and mechanics. He provides eight subsections of how these aesthetics can be achieved, although states that further subsections can also exist. These eight subsections are:

1. Sensation - Game as sense-pleasure
2. Fantasy - Game as make believe
3. Narrative - Game as drama
4. Challenge - Game as obstacle course
5. Fellowship - Game as social framework
6. Discovery - Game as uncharted territory
7. Expression - Game as self discovery
8. Submission - Game as pastime

This taxonomy should be applied to the game being produced during the requirements analysis, to identify the key goals of the game. These can then be defined as aesthetic models for gameplay, to aid the development of gameplay dynamics and mechanics that support these goals.

2.2.2. Gameplay

Gameplay itself is simply an extension of the game mechanics, as stated above. There is no one set definition of what constitutes a game’s gameplay. The reason for this difficulty of definition is because the concept of gameplay is so wide-ranging. Games designers will tend to view the gameplay from a completely different angle to the user for example, and even within these groups of people, gameplay can become very vague, or equally likely, very specific, extending over many areas. This is dependent on the individual’s ideas formed from exposure to many examples. As Andrew Rollings and Ernest Adams explain (11) ‘If you were to ask the question “What is gameplay?” most answers would attempt to explain by example…gameplay is so difficult to define because there is no single entity that we can point to.’

Sid Meier, creator of many successful game titles such as Civilization, once described gameplay as “a series of interesting choices”. This statement forms the basis of a more formal definition by Rollings and Adams (11) who define gameplay as, ‘One or more casually linked series of challenges in a simulated environment’. This definition seems to convey the general philosophy towards gameplay by many of those who are directly involved in games and appears to be the most relevant definition when thinking about the concept of a game’s gameplay.

When considering the gameplay in a project of this nature, we should contemplate the challenges being offered to the user in every aspect of the game. We can then make informed gameplay decisions, modifying the challenges where necessary by altering the game mechanics. This in turn will influence the gameplay and improve the overall game experience offered to the user.

2.2.3. Player Interaction

The way in which a player can interact with a game world directly influences the scope of information they receive pertaining to the game world. For
example, allowing the user to move anywhere within the boundaries of a virtual world will give them geographic information they could not obtain if their view was restricted to one location. Preece et Al (12) categorise an individual's interaction activities into four conceptual models, which we relate more specifically to the domain of an interactive game:

**Instructing** – This describes how users carry out their tasks through instructing the system what to do. In games, this typically involves controlling the actions of a virtual character (move forward, pick up, shoot).

**Conversing** – based on the idea of a person conversing with a system where the system acts as a dialog partner. This form of interaction is more common in games where the dialog of the characters in the virtual world can be controlled. Specific examples include the adventure game genre, where exploration through dialog is extensively utilised to drive the narrative forward.

**Manipulating and Navigating** – This conceptual model describes the activity of manipulating objects and navigating through virtual environments by exploiting users’ knowledge of how they would do this in the real world. Games utilise this concept in many ways, allowing users to control the virtual camera’s movement, zooming in/out on objects and manipulating objects of particular interest. Games simulating real life activities rely on the fact that their target audience will know how to interact with certain aspects of the game. In a snooker game, a user familiar with the real-life game will expect to be able to move around the table and manipulate a virtual cue object.

**Exploring and Browsing** - is centred on allowing the user to explore and browse information. In a game world users may need to examine certain aspects of the world carefully in order to gain the information needed to progress. In role-playing games, users are expected to explore the game world searching for specific artefacts they have been asked to retrieve. Similarly some games rely on conveying key task information by placing it in/on virtual representations of objects the user will relate to information gathering, such as a newspaper or computer.

These four conceptual models must be considered in the development of this project, identifying the categories of most relevance to a snooker game. We now look at a key area in controlling system interaction, the user interface, relating it to the domain of game development.

### 2.2.3.1. User Interface

Perhaps the most overlooked aspect of many non-commercial games, and occasionally commercial projects, a games interface is the virtual doorway into the game world and should be treated as a key element of the system. Perhaps the best analysis of a game’s interface can be credited to Bruce Shelley, a professional game designer and developer. In one article he details guidelines for successful development of games (13), acknowledging that ‘while the interface has little chance to dramatically enhance a game, there is a great risk...that poor interface design can do real harm’.
The amount of effort devoted to the design and implementation of a games interface in a commercial product can be staggering (2). How people interact with the system is now seen as a critical element in the development of software, as highlighted by the shift in industry trends described in section 2.1.3. However, some games are still let down by developers treating the interface as an afterthought to the gameplay. With this in mind, we will now look at the two main subsections of a game’s interface, the front-end and the control system, reflecting on the importance of each to the overall usability of a game.

**Front-End**

In a video game, a front-end is a set of GUI elements such as the menus, buttons and help text (2). Typically these allow the user to start new games, change the game options and quit the game, amongst other things.

A game menu is the method of navigation through the options a game has to offer, and should be accessible to all types of user. If the development of a front-end is deemed necessary for this project, we should ensure that the user can perform simple tasks such as starting a new game, without being confused by the visual design. If this cannot be achieved then the user will never be able to utilise all the features available in the game.

**Control System**

The control system, for any game, is an integral part of the system as a whole. A well designed control system allows for easy interaction with the game world, whether this relates to controlling a character in a first-person shooter or, in the case of this project, the control of an individual playing a game of snooker.

Control systems are briefly discussed in an article (14) by Ernest Adams, on the games website *Gamasutra*. He mentions that the control system of any sports game is constrained to the fact that the possible interaction activities are already defined by the rules of the game. This project will be no exception. For example, in snooker you must be able to select an angle and choose the power of the shot, so the controls must be tailored to incorporate this interaction.

The article (14) also mentions the difficulties of camera location and control. This is another crucial feature and must be representative of the choices you would make if playing the game in real life. For example, to give a better simulation of snooker it could be useful to be able to walk around and view the table at different angles rather than just showing one camera view from behind the cue. Both of these control issues should be carefully considered when designing the project.

The interaction considerations specific to this project, and the goal of realistically representing the game of snooker, will be further investigated during requirements gathering.
2.2.4. Player Enjoyment

Player enjoyment is not an easily quantifiable value. A game can be generalised as producing a certain level of enjoyment, but as with any form of media, it is always dependable on the opinion of those playing the game. This does not mean that the enjoyment of a game cannot be represented more precisely and there are many people who have formulated such representations as models in an attempt to do so.

One such model (15) has been developed by Sweetster and Wyeth from the University of Queensland, Australia. Their model attempts to adapt a number of concepts detailed in a paper by Csikszentmihalyi (16), in which he investigates into what makes human experience enjoyable. Sweetster and Wyeth adapt the results found by Csikszentmihalyi for optimal experience, or flow, to the games domain. The result is the GameFlow model, which consists of eight main elements:

- **Concentration** – Games should require concentration and the player should be able to concentrate on the game.
- **Challenge** – Games should be sufficiently challenging and match the player’s skill level.
- **Skills** – Games must support player skill development and mastery.
- **Control** – Players should feel a sense of control over their actions in the game.
- **Clear goals** – Games should provide the player with clear goals at appropriate times.
- **Feedback** – Players must receive appropriate feedback at appropriate times.
- **Immersion** – Players should experience deep but effortless involvement in the game.
- **Social** – Players should support and create opportunities for social interaction.

The model (15) can be considered in more detail than mentioned above, but these eight areas are what, according to Sweetster and Wyeth’s opinions and studies, a game should be judged upon to determine how enjoyable a game should be to a user. If a game fails in a number of these areas it could seriously damage the user’s enjoyment of the game.

Some of the eight elements are more crucial, dependent on the genre of game being developed. When designing and developing a snooker game, more emphasis needs to be placed on the elements of skill, control, feedback and social, as these are key elements for any game in the sports genre. It should also be noted that these elements will directly influence the game mechanics and so whenever the mechanics of the game are changed, we must think about how the overall value of the game experience has been affected.

2.2.5. Design and Usability Summary

The main points that can be taken out of the above research can be summarised by an excerpt from an article by Chuck Clanton (17), which states, ‘The
designer’s job is to create fun gameplay, and ensure that the game interface and game mechanic do not interfere’. Game designers above all else should ensure that the finished game has good gameplay, but not at the expense of other aspects of the game. It becomes easier to implement a component of the game, like a menu system, successfully if the design is thought about meticulously. In the case of the games front-end, a lot of developers overlook its importance, yet a badly designed menu system can detract from a game that is otherwise very well developed.

Another key point reiterated through many of the literature researched, would be the opinion that good game developers should always design games thinking of the user and working backwards. Some examples of this include the workshops by Marc LeBlanc (10) which state this point several times. Thinking of development this way means that at all stages the developer is considering what the user would enjoy, and likewise not enjoy. Developing each stage of the game with this in mind goes a long way to ensuring that the finished game will be enjoyable for the audience it was designed for. This opinion also reflects the importance of involving users in the process of game design and development, as this aids the discovery of exactly what features they want to see implemented in the completed system.

The alternative is to create a game from a concept and making the concept as good as it possibly could be. The main problem with this style lies in the realisation that if the concept being developed is not enjoyable, the user will not find the game enjoyable either. On a different level, making the game ‘better’ without considering the user’s enjoyment will more often result in features that the user does not want, or maybe even need, in the type of game being developed. This particular issue is termed feature creep and can occur in badly planned projects due to a lack of clearly defined development goals. These design realities should be kept in mind, whilst working towards a realistic development goal within the projects timeframe.

2.3. Graphics API

2.3.1. Which API should be used for a project such as this?

To develop a realistic snooker game we require an API that will provide a reliable, well developed environment. This will ensure the project development will not be hindered by problems that could be caused by a less developed API, simply due to these not having been refined over time by their developers. The API should uphold industry standards so that the project can be considered on a broader scale, with any issues in the development of the project being related to problems faced by those in commercial development and not just specific to the scope of this project.

It is crucial that the API has a good support network in terms of tutorials, its community and references (books, articles, etc.). The API chosen for development will be learnt from scratch by the developer and so having access to a variety of resources would aid in this learning process. This would also be of benefit when faced with common development problems related to this type of project. Choosing an API with a larger scope of support will in turn
limit the risks of choosing the API. A bad choice of API could seriously hinder the projects progress and, due to the time constraints on the project, the time required to switch development to utilising a different graphics API cannot be afforded.

The two most common graphics APIs available at present that fulfil the above criteria are the OpenGL API and Direct3D; the graphical branch of the Microsoft DirectX API. A third party API is also a possibility, but this would be a riskier strategy as support for any third party API would be substantially less than what is available for either OpenGL or Direct3D. In respect to learning an API that is an industry standard, a third party API would not be as useful as learning OpenGL or Direct3D, both of which are used to aid in professional games development.

Before deciding which of these two APIs would be most suitable for use in this project, we should look at both options carefully including their key strengths and weaknesses to evaluate which one will be better suited to the project requirements.

2.3.2. OpenGL vs. Direct3D

OpenGL is described by the official OpenGL website (18) as ‘the premier environment for developing portable, interactive 2D and 3D graphics applications’. It also states its advantages as being ‘Industry standard, stable, reliable and portable, evolving, scalable, easy to use and well documented’. These are all characteristics that would be ideal for use with this project.

Direct3D is described by Wikipedia.org (19) as ‘a 3D API…used to render three dimensional graphics in applications where performance is important, such as games’. It is a part of Microsoft’s DirectX API and has grown in stature recently as its recent iterations have looked to solve many of Direct3D’s previous problems.

There are many sources of reference available that compare OpenGL and Direct3D. However the majority of these are biased dependent on the preferred API of the author. We will therefore look at references that are as unbiased in their opinion as possible to ensure we take into account facts and not author preferences.

Through studying several articles, it seems that no clear advantage is gained by either API. OpenGL is clearly better in terms of portability over Direct3D. However, while this is an issue to mainstream developers, it holds no crucial value for the needs of this project.

Wikipedia (20) does however mention OpenGL’s extension mechanism as ‘...the most heavily disputed difference between the two APIs. OpenGL has a mechanism where drivers can advertise its own extensions to the API, thus introducing new functionality such as blend modes, new ways of transferring data to the GPU or different texture wrapping parameters. This allows new functionality to be exposed quickly...Direct3D on the other hand has a more consistent annually updated API, but can sometimes miss vendor specific features.’
This is also brought up in an article by Promit Roy (21) stating under Direct3D weaknesses; ‘DirectX is only updated every year or so, which is a little slow considering how fast the graphics industry moves.’ This, although a weakness for Direct3D, is not crucial to the needs of this project as the level of development being undertaken is sufficiently supported by Direct3D’s current API.

Overall, as expressed by Roy in 2002 (21), neither OpenGL nor Direct3D is clearly better than the other and this is still the case four years on. In fact emphasising this point further, Roy goes on to state that ‘if you are serious about games development, it is probably best to learn both Direct3D and OpenGL at some point’. OpenGL has more widespread use currently in graphical development as well as in the games industry. Its better portability features and slightly less complicated approach for programmers new to OpenGL, keeping the programmer at arms reach from the hardware, make it a better choice for our particular needs. OpenGL will therefore be the API utilised in this project. To get a background of OpenGL we look briefly at its history in the next section.

2.3.3. OpenGL – A Brief History

Although the relevance of looking at OpenGL’s history is debatable, it is often worthwhile to look at the origins of a standard in this way to gain a better perspective of its strengths and weaknesses, before starting development. We shall therefore look briefly at the milestones in OpenGL’s history.

Fifteen years ago, developing software that could function with a wide range of graphics hardware, with all their different interfaces, was a major problem. Individual teams of programmers tended to develop their interfaces separately, producing plenty of duplicated code, and this separate development habit was hindering the growth of the graphics industry. Silicon Graphics, Inc. (SGI), being the world leader in 3D graphics at the time, were the first to consider this a major problem and set about trying to standardise access to hardware. Their solution was OpenGL, a multi-purpose, platform independent graphics API that pushed the development responsibility of hardware interface programs to the hardware manufacturers. This ensured the software developers could concentrate on developing software (22).

Since 1992, OpenGL development and specification has been administered by the OpenGL Architecture Review Board (ARB). The ARB is made up of a collection of major graphics vendors and numerous other industry leaders from many backgrounds, all with a vested interest in developing a consistent and accessible API. These companies include 3DLabs, ATI, Dell, Hewlett-Packard, IBM, Intel, NVIDIA and SGI to name but a few. Microsoft was one of the founding members of OpenGL, but left in March 2003 to work on Direct3D, now the leading competitor to OpenGL.

OpenGL intentionally provides only low-level rendering routines, allowing the programmer a great deal of flexibility and control, but the provided routines can easily be used to construct high-level rendering and modelling libraries. The OpenGL standard also allows individual vendors to add features and functionality through extensions as new technology is developed. These, if
adopted by a number of other vendors or deemed important enough by the ARB, may be promoted to the core OpenGL specification.

OpenGL, currently reaching version 1.5 with version 2.0 being proposed currently, provides the programmer with an interface to graphics hardware. Many games such as id Software’s Doom 3 and recently released Half Life 2 by Valve Software use OpenGL for their core graphics rendering engine.

At this stage it is hard to propose exact features of OpenGL that will be utilised in development, having never previously developed a project in OpenGL, and this will be unclear until the development process starts. However, looking towards the design and development processes we can identify areas that will need to be researched throughout the development process. One area that will need to be researched is the OpenGL extension implementation, as this may be needed for windows compatibility for any features beyond OpenGL 1.1 should they be required in implementation(16), as Microsoft, since leaving the ARB in 2003, have not kept up with the latest OpenGL specification. This is mainly due to Microsoft’s Direct3D being in direct competition with OpenGL.

It is important to learn how to develop very basic graphical structures and how to put these together to form more complex structures. Good resources for this include ‘Beginning OpenGL game programming’(22) as well as several well supported online tutorials (23) (24), which detail how to develop very simple applications, right through to more complex OpenGL techniques.

2.4. Game Physics

2.4.1. Why does a game need a Physics Engine?

We experience physics everyday in our lives. Aspects of real-world physics such as gravity and friction are more or less taken for granted. It is because of this that games, and in particular games that relate to real-life, include a representation of physics. Physics ultimately help pull a game into reality (25) (26).

Considering this project more specifically, the game will be a representation of a real-world activity; the game of snooker. A video game representation of snooker has various physics to consider, most notably collisions, gravity and friction. The aim of the project is to design and implement a snooker game and this encompasses many areas. If we were to develop a physics engine for a game from the ground up, this would take up a substantial amount of development time. Therefore it is more feasible to use a pre-built physics engine that is freely available and encompasses the functionality required for the project.

The best option will be to use the Open Dynamics Engine (ODE). This is due to the fact that ODE is a freely available, professionally used engine unlike alternatives, such as the Havok Game Dynamics engine, which are only commercially available. It has an easy to use C++ API, which is the language to be developed in (see section 5.1), and the physics engine itself is capable of
implementing the physics required in the game. There is also good support available for this engine internally with the university, as a current research student has used the ODE engine previously, should further advice be required. All of the above factors make this physics engine the most practical choice to use for this project.

Although using an existing physics engine, to get the most out of the system we should understand the basic underlying physics being used (2). The following sections look closer at collision detection and friction, which will be the most important aspects with regards to the game physics in this project.

2.4.2. Collision Detection

In a game world, objects will pass through each other without hesitation. Solidness is the property that must be implemented in a game to stop this happening and this is implemented through collision detection and resolution (2). This is a crucial property for my particular choice of game, as in snooker collisions are an essential aspect of the game. Without implementing this solidness into the virtual world of my game, snooker cannot realistically be simulated.

Collision detection determines if and when two objects collide (2). Since it would not be enough to merely detect the collision, collision resolution works out where each object should be once a collision is detected (2). Calculating how the objects move after the collision is the job of the physics and needs to consider many real-world elements such as friction. Collision detection is fundamentally a kinematic problem, involving the positional relationship of objects in the game world, whereas collision resolution is a more dynamic problem, in that it involves predicting behaviour according to physical laws (26).

Determining if and when two objects collide is not as simple as it may initially appear. In this project, the fact that the only collisions will either be between two spheres (two balls colliding), or between a sphere and the game boundary (the ball hitting a table cushion) makes the geometry of the collisions simpler than if they were a more complex geometric shape, such as a modelled game character. Collision detection is a fairly costly process, as fundamentally every object in the game world should be tested against every other object for the possibility of a collision, which results in $O(n)$ time complexity. Because of difficulties such as this, many strategies have been devised to perform collision detection in real-time during gameplay.

There are a few techniques available for detecting a collision. In ODE, a collision detection engine is supplied, to which information about the shape of each object is given. Then at each time-step it figures out which physical bodies are in collision with one another and passes the resulting contact point information to the user (27). The user in turn creates contact joints between objects. ODE’s collision detection is optional and alternatives can be used as long as it supplies the correct kind of contact information (27).

Two alternative techniques that are popular in game development are overlap testing and intersection testing (2). Overlap testing basically detects whether a
collision has already occurred, whereas intersection testing predicts if a collision will occur in the future (2). The next section will look briefly at these techniques and their limitations.

Overlap Testing

This is the most commonly used collision detection technique, but also displays the most error. The basic concept is that at every step, each pair of objects, for example two spheres, is tested to clarify whether or not they overlap with one another. If they do, then they are in collision and results are returned informing of the time of collision and the collision normal vector. To calculate exactly when the collision took place, the two objects must be moved back in time to the last simulation step when they were not in collision (2). Using a technique called bisection the simulation is moved forwards and backwards by half of the last simulation step, to discover the exact collision time of the two objects. This is shown in figure 3 below, which shows how bisection determines the exact collision time (2).

![Fig 2.2. Using Overlap testing, it has been detected that the moving object A collides with the stationary object B at time $t_1$. To find the exact time before intersection, five iterations of bisection are performed in order to converge on the time right before collision (2).](image)

Overlap testing is unfortunately limited, as it does not work very well for very fast moving objects. I.e. A bullet fired at a window (2). Since the bullet is very small and is travelling very fast, an overlap with a thin window object is unlikely to ever occur during a single simulation step. This results in the bullet passing directly through the window, with no collision being detected. For overlap testing to work, the speed of the fastest moving object multiplied by the time step must be less than the size of the smallest object it can collide with in the scene (2).

Alternatively, the size of the simulation step could be decreased, but doing so may result in the simulation running at hundreds of frames per second, and this is impractical due to the subsequent detriment of runtime performance.
Intersection Testing

Intersection testing differs to overlap testing, as rather than waiting for the collision to occur it attempts to predict if collisions are going to happen at a simulation step in the future. Once predicted the simulation can then be moved forward carefully to the time of impact. This process is often more accurate and efficient than overlap testing (2). Intersection testing assesses the geometry of an object swept in a specific direction of travel against all other objects’ swept geometry. The object must be extruded, irrespective of its geometry, over the distance of travel during each simulation step. For example, a sphere would become a capsule and a square would become a rectangle. This extruded geometry is then evaluated against other objects’ extruded geometry to find any collisions.

In the specific case of sphere-to-sphere collision, the exact point of impact is calculated by a direct formula (see equation 1) (2).

\[
 t = \frac{-(A.B) - \sqrt{(A.B)^2 - B^2(A^2 - (r_p + r_q)^2)}}{B^2}
\]

Where:

\[
 A = P_1 - Q_1 \\
 B = (P_2 - P_1) - (Q_2 - Q_1)
\]

**Equation 1. General formula to show point of impact of two spheres in intersection testing**

A sphere of radius \(r_p\) moving from point \(P_1\) at time \(t=0\) to point \(P_2\) at time \(t=1\) collides at time \(t\) with another sphere of radius \(r_q\) moving from point \(Q_1\) to \(Q_2\) (2). If \(t\) falls into the interval \((0,1)\) then a collision is detected (see Figure 2.3).

The limitations on intersection testing include using this form of testing for networked games. This is due to the fact that future predictions rely on knowing the exact state of the game world at the time of each test. In a network game, the state is not always coherent and so collisions can be calculated incorrectly, including the prediction of collisions that will not happen at any point in the future.
Further Considerations

Further collision detection algorithms do exist. One such example, detailed in a paper by Moore and Wilhelms (26), goes into further possible methods for both collision detection and response. Papers such as this are useful for background understanding of potential high-level solutions, but stretch beyond the scope of what will be implemented in this project and so will not be detailed here.

From research, it appears that using the detection and response provided with the ODE physics engine is the best implementation to utilise. Therefore we should allow development time to investigate the implementation of ODE’s detection mechanism and how it will integrate with the system. However, as our knowledge of this subject grows through development research, this could be reviewed and we could opt to utilise another method if it becomes necessary.

2.4.3. Collision Resolution

Collision resolution is the response to a detected collision. In a video game environment the situation, and what two objects caused the collision, will affect the type of resolution required. Coming back to a previous example described in section 2.4.2., when a bullet object collides with a window object, the window should smash, with perhaps a sound effect on impact. Another example would be if a rocket is fired and hits a game character. The rocket should disappear, an explosion with a sound effect should be created at the impact point and damage should be inflicted on the game character that the rocket hit (2).

Considering the game to be developed for this project, the main type of collision will be between two snooker balls or a ball and a table boundary (cushion). The position of the balls at the time of collision must be calculated, to place them in the correct location at the time of impact (2). Resulting linear and angular velocities must also be imparted onto the balls (26) and perhaps a sound effect triggered upon collision.

As collision resolution varies depending on the situation and objects taking part in the collision, a procedure has been suggested for the resolution (2). This procedure has three parts:

- **Prologue** – When a collision begins, determine whether or not the collision needs to be ignored or if some action should take place.
- **Collision** – The objects are placed at the point of impact and new velocities assigned to them using physics.
- **Epilogue** – Any post-collision effects must be triggered (2), such as playing a sound effect or inflicting damage. This can be done by sending a collision epilogue event notification to each object, with the object determining what is triggered.

Events, such as sound effects, can take place in either the prologue or the epilogue, depending on the game design and circumstance (2).
The collision part of this procedure will depend on what type of collision detection is used. Taking the methods explained in section 2.4.2. as examples, intersection testing resolution is fairly straightforward, as the two objects never actually penetrate. We would only need to extract the collision normal vector at the time of collision and then calculate the new velocities on each object (2).

Overlap testing is slightly more complicated as the objects have already intersected. The collision normal is therefore obtained by finding the position of each object immediately before collision. Using the two closest points on each surface, the collision normal can then be constructed. Once this has been extracted we can then extract the penetration depth, using an algorithm such as Gilbert-Johnson-Keerthi (GJK), and move the objects to a penetration depth of zero (2). Only then can we compute each objects new velocity using physics.

We should consider how our application could use the procedures suggested and what actions need to be triggered upon the occurrence of specific types of collision in the game. The basic collision of ball-on-ball should be relatively straightforward to detect and resolve, but to do this realistically will be more difficult as the collision will be simulating real-life physics. This will be where the physics engine should take the brunt of the leg-work, so that we do not have to implement the underlying physics into the system, instead concentrating on fine-tuning specific ODE variables to give an accurate physical representation.

2.5. Conclusions

Many of the issues researched in this literature survey were to gain an understanding of areas that need consideration before any sort of development can take place. Design issues are numerous and from this review it is clear that certain aspects need be prioritised over others to gain a well balanced system. For example, a strong emphasis should be placed on the user interface of the game, especially in relation to the control system, as this is how the user will directly manipulate the game itself. Although relatively simple to implement, this should not be overlooked, as given attention and through obtaining user feedback, it is a part of the system that should be relatively easy to develop successfully. If this were to be overlooked, the user may not even come into contact with the other elements of the game and this would be a basic failure of the system.

The survey also provides a better understanding of the features of OpenGL and aspects of game physics, in particular collision detection, since this is the most relevant attribute in a game of this nature. As the project progresses, understanding of these areas will broaden as the features of OpenGL and ODE are utilised throughout the development stage. It will be important to review these two attributes throughout development, as together they make up the visual and physical aspects of the game and so directly affect the games realism.
Chapter 3

Requirements Analysis and Specification

3.1. Introduction

The Literature Survey has highlighted and discussed the main issues that should be considered throughout the design and implementation of this project. It has become clear that developing a successful game for any genre is more than simply a case of creating a graphically impressive world or a realistic physical simulation. A number of factors are required (graphics, game mechanics, UI, physics) that in combination can generate an exceptional interaction experience.

Due to the nature of the problem domain, it would be unwise to only build our requirements on the theories discussed in the Literature Survey. Game design is an interactive and practical field. We shall therefore evaluate several existing products, before further analysing the problems of realistic interactivity in this project. Unlike productivity software, in which the user brings their goals to the application, games applications bring the goals to the user. It is therefore important to analyse the goals of the game being produced and its associated rules. Consequently, the game of snooker and the rules required to produce an accurate representation of the sport will be discussed.

It is important to note that the process of requirements analysis should always consider the potential system stakeholders. For commercial game development projects this would be done on a very large scale, as stakeholders are anyone who could ever come into contact with the game, encompassing much of the general public. Stakeholder analysis on this scale would be very time consuming and is unfeasible to consider allocating this amount of time to one section of requirements gathering within the constraints on the project. However, it would be unwise to ignore stakeholder analysis completely and so it is briefly considered in section 3.3.
3.2. Evaluation of Existing Products

We will look briefly at two existing snooker/pool games. The first is the recently released World Snooker Championship 2005 (WSC), a commercially available snooker game developed by Blade Interactive. The second is a freeware pool game, FooBilliard, available online (28). Discussed here are the main findings of the evaluations, although they can be found in full in Appendix A.

3.2.1. Main Observations

Both of the games evaluated implemented the features a user would expect to see in a game of this type, such as realistic ball collisions, ball movement and accurate rule implementation. The clear differentiating factor between the two games, as expected when comparing commercial and freeware products, is the number of features available to the user.

WSC, as a commercial product, has been designed to provide many game options to the user, rather than just the single mode of play FooBilliard offers. Although WSC does contain many more modes and features, many of these would be left unexplored by the user, as they are simply adding depth to the main focus of the game. Although FooBilliard only offers the one game mode, it is implemented very capably ensuring the lack of any extra features does not detract from what is present.

It is worthwhile noting that neither game is clearly more competent when compared in terms of ball physics. This leads to the assumption that it is better to implement the key features of the game well, concentrating efforts on ensuring these are of a high standard, than implementing many features that the user may never utilise. This point was proved further during the stakeholder analysis (see section 3.3).

Both games let themselves down in what we have found to be one of the most overlooked areas. Neither game has a suitable front-end user interface. WSC utilises a rather complicated menu system that is simply impractical for navigation between options, while FooBilliard lacks any menu or help structure at all. This could be explained away by the unnecessary need for such a front-end due to the lack of customisable options and game modes present in FooBilliard, but some kind of explanation of the controls would greatly enhance the usability of the product.

3.2.2. Conclusions from Evaluations

The evaluations have provided the following useful considerations for what this project should aim to achieve:

• The project should not focus on developing large numbers of unnecessary features, but should focus on implementing key features as accurately as possible. Although more features add depth to a game, they will not be of substantial value if the main features are not in place.
• The evaluation has reaffirmed the points raised in the Literature Survey (see Section 2.2.3.1.). Any user interface implemented should be easy to use and not overcomplicated. If, like FooBilliard, a front-end menu system is not utilised, clear instructions for how to use the game should be accessible, perhaps in the form of a user manual or help screen.
• The difficulty of the game is dependent on how the control and aiming systems are designed. WSC employs an aiming system which can make the game relatively easy. In comparison, FooBilliard simply provides a line down the path of the cue ball which makes the game difficult in comparison. This is a consideration that should be examined further in the design of the project.
• A scoring system will be required to keep track of player scores and to indicate whose turn it is.
• The graphics do not need to be a key implementation priority in this project. The aim should be to achieve realistic visuals, but not to overcomplicate them, as other features such as realistic ball physics should take priority.

3.3. Stakeholder Requirements Analysis

Any video game can potentially be used by anyone who wishes to play it. Therefore, stakeholders for our system could be any member of the general public who chooses to use the game. This can be broken down into two subcategories; users who are familiar with the game of snooker and users who are not.

Stakeholders from each subcategory were asked to fill out questionnaires, for us to gain information relating to their opinions on what features were imperative to a video game representation of snooker. The results of these questionnaires can be found in Appendix B. These results must be considered in light of only twelve stakeholders being questioned. To gain a solid understanding of stakeholder priorities, further analysis would need to be conducted.

From analysing these results, it was apparent that features which provide better system functionality are favoured to features that simply provide depth to the game experience. Quality of graphics, sound effects and number of game modes were all considered features of a lower priority by stakeholders. It is interesting to note that these results reflect the current trend of the industry, with game visuals now considered less of a priority to consumers than the quality of the interaction provided. Features that have been confirmed as the highest implementation priorities include the ball speed, rule implementation, scoring system and power indicator. The results especially highlight the importance of an easy to use control interface, with this being considered a maximum priority by eleven of the twelve stakeholders questioned. We must realise the importance of user participation in the design and implementation of this aspect of the system, as a well developed control system will enhance the entire system’s usability.
Stakeholders also identified features not previously identified as important to the system. The feature most feasible of those suggested, in relation to this project, was the possibility to adjust the angle of the cue before playing a shot. This is a feature that would give the user further choice in the type of shot they could play. Its implementation in the system should be considered, although perhaps only as an optional inclusion due to its relative importance when compared with other system functionality.

3.4. Interactivity Requirements Analysis

As discussed in the Literature Survey (Section 2.2.3.) interactivity is an important consideration in any game development project. This is especially evident in this project, where the subject of the game is a real-world activity. We need to therefore consider what important elements of interactivity need to be adapted from the real game to its virtual representation. Identified from research conducted up to this point, there are a number of interactivity considerations that should be assessed.

In a real game of snooker, a player can view the state of play from a multitude of angles before deciding which shot to attempt. This aspect of the virtual world needs to be addressed so that the user can view the virtual table from a number of different angles. How many angles are required to provide a realistic experience is ultimately a design decision, but the game should provide the user with an appropriate number of views to ensure that they can come to a clear decision as to what shot they wish to attempt.

The control system is another consideration. In real life, the power of the shot is determined by the player hitting the cue ball with a cue. This sense of force needs to be conveyed in the game, either visually using a power indicator or, like several existing products, by using the mouse to simulate the cue movement. The latter is usually controlled by the speed of mouse movement; the faster the mouse is moved forward the larger the force applied to the cue ball.

Being able to select the direction the cue ball will travel in is the most important interaction choice a player has to make when playing snooker in real life. If the correct angle is chosen, the shot will be a success. To represent this virtually, we should look at implementing an aiming system to show the user where the cue ball is currently being aimed. The specific detail of this aiming system should be considered carefully in the design.

One challenge raised by Preece (12) is the decision to utilise realism or abstraction when designing a system that contains real world entities; should these objects be designed to give the illusion of behaving and looking like its real-world counterpart, or as a visual abstraction of the object being represented? In the case of a snooker game, it is crucial that the objects represented both look and behave like their real life counterparts, as we are trying to achieve an interactive simulation of a real life activity. If the objects did not behave in an expected manner, users with knowledge of the game would be left confused. The game must therefore represent snooker as accurately as possible, through the interaction methods employed.
3.5. MDA Framework

The Literature Survey outlined an approach (the MDA Framework) that aids in the formalism of a game's aesthetic goals, and the subsequent game dynamics and mechanics that can be defined to achieve these goals. We have applied this MDA framework (9) to our project to classify the system goals. From these we hope to identify specific game dynamics and mechanics relevant to the game being developed.

This project encompasses the aesthetic categories of Sensation, Competition, Challenge and Submission. These are the high level aesthetic goals of our project and are described below.

- **Competition** - The game succeeds in this criterion when the players are emotionally invested in defeating each other. The system requires clear feedback about who is winning to achieve this quality. A failure of this criterion would be if a player feels that they cannot win, or there is an inability to measure their progress in the game.

- **Sensation** - is achieved by allowing the completion of set objectives within the game to give the user a sense of pleasure. One evident goal of the game is to pot one or more balls, in the order specified by the rules of snooker (29), within a player’s turn. The relative success of this can be measured by the player’s current ‘break’; the total number of points scored in one turn. This information should be displayed onscreen for the user to quantify their success. The system fails at this aesthetic if certain objectives cannot be completed, or if the game allows an objective to be completed with excessive ease, as very little pleasure will then be attained through its achievement.

- **Challenge** - can be achieved by the level of difficulty relating to the main game objectives. In this project, this is how difficult it is to pot a ball, which is primarily controlled by the aiming system. The aiming system must be designed to provide a suitable level of challenge. Failure of this would be if the user feels the difficulty of the game is too easy or too hard.

- **Submission** – The game must ensure the user wants to improve and play the game more frequently. The game must therefore provide scope for players to learn and hone their skills. The game fails at this if the player feels they cannot improve their skills, as they will lose the motivation to play the game.

These aesthetic goals and their relative success and failure states should be reflected in the requirements specification.

3.6. Rule Centred Requirements Analysis

The rules of snooker have placed a number of specific requirements on the project, as the project must represent the real game as accurately as possible.

The rules that shall be conformed to in this project have been taken from the official website of the World Professional Billiards and Snooker Association WPBSA (29). A full explanation of the rules most relevant to this project can
be found in Appendix C. However, some of the rules will need to be considered as lower priority than others due to the time constraints placed on this project. Low priority rules will be those which are not critical to the game as they would not affect the overall playability.

An example where such a compromise will be made is the ‘miss’ rule. This is a complex rule which rarely comes into play, involving the optional replacement of balls when a foul has been played (see Appendix C, section 3.13 for a full explanation of this rule). This rule would not only be complex to implement due to the considerations of recording ball locations before every shot, but would not affect how the rest of the game could be played if it was missing from the final system. Therefore, this rule must be considered as a low priority compared to rules such as re-spotting colours and accurate scoring, both of which are central to the game of snooker.

All rules have been analysed in this way, given priorities based, not just on the affect on the system, but also in respect to stakeholder analysis to ensure that features considered important by potential users are given higher priority than those that are considered less important. A more formal specification of these requirements can be found in section 3.6 as well as each requirement’s specific level of priority.

3.6. System Considerations

The game will be developed on a relatively high-end PC (Pentium 4, 3.20 GHz), with high-end graphics capabilities. It is important to consider that not all users will have this level of system specification and the game should be developed with this in mind where possible. The constraints on the reality of this being achieved will become more apparent when the efficiency of the physics engine has been investigated in the implementation. Until then, we will set the goal of ensuring the system runs on medium to high-end PCs, with high-end graphics capabilities. This requirement will be revised during the implementation and testing process if necessary.

3.7. Requirements Specification

3.7.1. Requirements Structure

The requirements specification follows the standards outlined by Sommerville (1) and Preece et Al (12). The two main categories of requirements outlined by Sommerville are Functional and Non-functional requirements. Preece et al suggests that referring to all requirements that are not functional as “non-functional” is too vague when designing for interactive systems and should be refined into further subcategories. Therefore four categories of requirements will be detailed:

1. **Functional Requirements**
   The functional requirements for a system describe the functionality or services that the system is expected to provide.

2. **Non-Functional Requirements**
These relate to the constraints on the services or functions offered by the system. This includes constraints on the system’s performance and on the development process.

3. **Usability Requirements**

These capture the usability goals and associated measures for a particular product.

4. **User Requirements**

Relate to the skills and characteristics of the targeted user group.

Usability and user requirements are two refined subcategories of the non-functional requirements that are of specific interest in this project. Usability requirements encompass the goals of the systems user interface, which is an area we have discussed the importance of in the Literature Survey (see section 2.2.3.1.). The user requirements will cover how the system should be designed with different types of stakeholders in mind.

### 3.7.2. Requirements Format

As mentioned previously in this chapter, due to the limited timescale for development, priorities will need to be placed on the requirements.

Requirements that are considered a high priority and essential to the system, will be written in the context of “The system must...”.

A low priority requirement is deemed non-essential to the system and will use phrasing such as “The system should...”.

### 3.7.3. Functional Requirements

1. **The graphics must realistically represent the equipment needed to play a game of snooker**

   1.1. A graphical representation of a snooker table must be present with dimensions relative to those of a physical snooker table, as specified by the WPBSA regulations (29).

   1.2. The balls must be positioned on the table in accordance to the standards laid out by the WPBSA (29).

   1.3. The balls should be realistically represented in the game.

   **Justification:** The user must be able to instantly recognise that the game being represented is snooker. The graphics must therefore be of a suitable standard, accurately representing all of the necessary equipment, to achieve this.

   **Source:** Gathered from Literature Survey.

2. **The game must feature basic ball physics that realistically simulate how the balls would move in a real-world game of snooker.**

   2.1. The balls must realistically collide with the cushions and other balls on the table.

      2.1.1. The angle at which a ball rebounds off another object must be accurate when compared to the real game.

      2.1.2. The speed at which a ball rebounds off another object must be accurate when compared to the real game.

   2.2. The balls movement must conform to real world constraints, such as gravity and friction.
**Justification:** As the game is a representation of a real-life game of snooker, real-world physics must be present to give an accurate representation of the game.

**Source:** Gathered from Literature Survey and Stakeholder Analysis.

3. **The system must conform to the rules that are critical to playing a game of snooker.**
   3.1. The system must recognise what ball should be hit/potted next.
   3.2. The system should request the nomination of a colour when a red has been successfully potted.
   3.3. The system must re-spot the cue ball when it is potted.
   3.4. The system must correctly identify when a coloured ball should or should not be re-spotted.
      3.4.1. The system must identify the correct position a ball is to be re-spotted to.
      3.4.2. The system must be able to cope with other balls occupying the location of a re-spot, taking appropriate action in relation to the rules.
   3.5. The system should recognise when the end of a game has been reached.
   3.6. A player should have the option of conceding the game at any point.
   3.7. The system should identify when the ‘miss’ rule should be applied (as detailed in Appendix C, Section 3.13.).
   3.8. The system should allow a player to request that the opponent is put back into play if a foul has been committed.

**Justification:** The rules should be implemented as accurately as possible, to give the best possible representation of the real-life game of snooker.

**Source:** Gathered from Rule Centred Analysis.

4. **The system should allow for movement of the cue ball in relevant circumstances.**
   4.1. The cue ball movement should be limited to the boundaries of the D marked on the table.
   4.2. The cue ball should not be able to collide with any other ball when being moved within the D.

**Justification:** Cue ball movement is available in certain circumstances in snooker (at the start of a frame or when the cue ball is potted), enabling the player to line up the best choice of shot within the boundaries of the D. Without this implementation, users who do foul could end up gaining an unfair advantage.

**Source:** Gathered from Rule Centred Analysis.

5. **The system must provide an accurate scoring system.**
   5.1. The system must identify when a successful pot has been achieved and update the player’s score appropriately.
      5.1.1. The system should record and display the current break achieved by the player currently at the table.
   5.2. The system must identify when a foul has been made and update the score appropriately.
   5.3. Both players’ scores must be visible onscreen.

**Justification:** A scoring system should be provided so that the users can compete with one another effectively. Without a scoring system the game would become a simulation with no way for players to judge progress against one another, losing the competitive aspect of the game.
6. **The game must allow the user to select the direction in which the cue ball will be struck.**

6.1. The system must provide a method of showing the user where the ball is being aimed.

**Justification:** The first priority of someone playing snooker is to select the direction or angle they wish to play their shot. This is an important interaction to represent in the game because essentially this feature is the most apparent interaction decision a player has to make. Providing an aiming system helps kerb the difficulty of the game. The exact type of aiming system should be considered in the design of the system to ensure the game developed is not excessively easy or difficult.

**Source:** Gathered from MDA Framework, Evaluation of Existing Systems and Interactivity Requirements Analysis.

7. **The game must provide a method of controlling the power with which a shot will be played.**

7.1. Power control must be represented visually or by utilising another appropriate method so that the user knows the relative amount of power with which they are playing the shot.

7.2. Increasing the power control must increase the force with which the shot is played. Similarly, decreasing the power control must decrease the force with which the shot is played.

**Justification:** Controlling the power of the shot being played allows the user to interact with the game as they would in real life, adding more power to hit the ball with more speed. Playing each shot with the same amount of power would severely detract from the overall experience, as it would limit the possibilities available to the user.

**Source:** Gathered from Stakeholder Analysis.

8. **The system must provide a number of camera viewpoints of the game world.**

8.1. The system must provide a suitable number of viewpoints to give a realistic interaction experience when compared to the real game of snooker.

**Justification:** Providing a number of viewpoints will allow the user to interact with the system much like they would in the real world, looking at the table from a number of perspectives before choosing what shot to attempt. Without appropriate viewpoints, the interaction experience could be affected adversely.

**Source:** Gathered from Literature Survey and Interactivity Requirements Analysis.

9. **The system must ensure it is clear which player’s turn it is.**

**Justification:** Snooker is a two player game, and so there should be some sort of prompt, whether that is visual or through utilising another method, to show the users who should take the next shot.

**Source:** Gathered from Rule Centred Analysis.

10. **The system must provide spin effects to be applied to the cue ball.**

10.1. The system should notify the user as to what spin is being applied, whether visually or by another method.

**Justification:** Spin effects allow for greater flexibility in the type of shots that can be played. Therefore by allowing the user to use spin effects, certain shots that would have been otherwise unachievable,
become possible. This enhances both the number of options available and realism of the game.

**Source:** Gathered from Evaluation of Existing Systems and Stakeholder Analysis.

11. **The system should provide a method to control the vertical angle at which the cue strikes the cue ball.**
   11.1. The system should notify the user as to what vertical angle is being applied, whether visually or by another method.
   **Justification:** The vertical angle at which the cue ball is hit effects how the shot is played. Advanced spin shots can then be played that could not be achieved without this feature. This, similarly to the addition of spin effects, enhances the number of options available and realism of the game.
   **Source:** Gathered from Stakeholder Analysis.

12. **The system should utilise sound to enhance the game.**
   12.1. Sound effects should be applied upon appropriate onscreen events, such as two balls colliding.
   **Justification:** Sound effects, although not a necessity, would add to the overall depth and realism of the game.
   **Source:** Gathered from Evaluation of Existing Systems.

13. **The system should provide a help facility for users.**
   13.1. The help facility should provide information relating to the game rules and controls.
   **Justification:** A help facility should be provided to ensure that users unfamiliar with the game rules or controls can easily learn to use the system. This would make the game accessible to stakeholders with no knowledge of the system or the game of snooker.
   **Source:** Gathered from Literature Review and Evaluation of Existing Systems.

### 3.7.4. Non-Functional Requirements

1. **The system should run on medium to high-end specification PCs with high-end graphics capabilities.**
   **Justification:** The game should aim to run competently on the majority medium to high-end specification computers. The system may run on systems with lower specification but the system will not be developed with this in mind.
   **Source:** Gathered from System Considerations.

2. **Output from the system must be accurate.**
   2.1. The scoring system output should be accurately portrayed in accordance with the rules of snooker.
   2.2. The system must not provide any unexpected output
   **Justification:** Preece et Al (12) state that users can become frustrated by unexpected output, with this usually a result of bad system design. The impact of this frustration can be as drastic as the user simply abandoning the application altogether. We must ensure that the output does not provide the user a reason to become frustrated.
   **Source:** Gathered from Literature Survey.

3. **The system must be reliable and consistent.**
   3.1. The system must not fail at any point.
Justification: Unexpected occurrences such as a system crash will destroy the user’s confidence in the product. It therefore must be ensured that the system will not enter a state of deadlock or failure.
Source: Gathered from Literature Survey.

4. The project must adhere to the standard project constraints placed on it by the University.
4.1. The project must be completed and handed in by the 8th May 2006.
Justification: The project must account for external limitations placed on it such as the deadline for the project.
Source: Gathered from Project Guidelines.

3.7.5. Usability Requirements

1. The user interface must be easy to understand and use.
1.1. The Control system employed must be accessible, logical and uncomplicated, and must not hinder the way in which the system is used.
1.2. Menus should be clear and easy to navigate.
Justification: A systems user interface, as discussed in the Literature Survey and the Requirements Analysis, is an area that if not given due consideration can seriously affect the usability of the product. It is better for the user interface to be less complicated and easy to use, than sophisticated and difficult to use.
Sommerville (1) states that you should apply Shneiderman’s user interface design principles to the design of a systems user interface. These include User Familiarity and User Diversity. These principles will be discussed further in the design chapter of this document (see section 4.6.1.).
Source: Gathered from Literature Survey and Evaluation of Existing Systems.

2. The interface should be highly learnable, so that users with no knowledge of the game of snooker can learn how to use the system.
Justification: From the Stakeholder Analysis, the system has two main categories of users; those who know are familiar with the game of snooker and those who are not. The system must allow both sets of users to become skilled at using the game. This could be done by utilising graphical prompts or an informative help section.
Source: Gathered from Stakeholder Analysis and Evaluation of Existing Systems.

3. The interface must be memorable to frequent users of the system.
Justification: The interface, in particular the control system, should be easy to remember so that frequent users can use the system without the need to browse through help facilities or user manuals.
Preece et al (12) state ten usability principles developed by Jakob Nielsen, with this requirement covered under principle 8 ‘flexibility and efficiency of use’.
Source: Gathered from Literature Survey.

4. The system should be accessible by one or more users.
Justification: Snooker, as a standard, is played by two players. The system being developed could however be played by a single user, playing against themselves, or by numerous user, with one or more on a
team. We must ensure that the system allows accessibility to users when the system is used in these scenarios.

Source: Gathered from Rule Centred Analysis and Stakeholder Analysis.

5. **Output from the system, whether visual or otherwise, must be clear and unambiguous.**
   
   Justification: Output, such as visual prompts and the scores, must be clear in the message or function they are conveying to the user. If this message is not clear, consideration should be made as to whether the output adds anything to the usability of the system.
   
   Source: Gathered from the Literature Survey.

6. **The system should operate at an efficient frame rate so that it can be used by computers of an appropriate specification.**

   6.1. The code should be kept efficient so that the frame rate does not run at a noticeably slower speed on medium to high-end specification computers.

   Justification: The game should run on computers with specification as detailed in non-functional requirement 1. The frame rate chosen should allow for this specification computer to use the system without compromising the usability of the game.

   Source: Gathered from Literature Survey.

3.7.6. User Requirements

1. **The system must support all types of user, irrespective of the level of knowledge they possess relating to the game of snooker.**

   Justification: As discussed in the stakeholder analysis there are two main branches of stakeholders; those with and those without knowledge of the game of snooker. It is important to provide support to those users that are unfamiliar with the rules of snooker, so that a large number of stakeholders are not at a disadvantage to other stakeholders.

   Source: Gathered from Stakeholder Analysis.

2. **The system should exhibit a sensible level of difficulty, allowing the facility for users to improve their skills as they use the system more frequently.**

   Justification: All users want their competency level to grow the more frequently they use a system. This is especially so in relation to games, as the user will not want to keep playing if they cannot get better through practice. The system should therefore provide facilities for improvement. An example of this is to provide an aiming system that the user can adapt to and handle more effectively the more often they use it. If users cannot improve, they will probably choose to no longer play the game.

   Source: Gathered from Literature Survey, MDA Framework and Evaluation of Existing Systems.

3. **The system must ensure users can judge their progress in the game, in order to satisfy the competitive aspect of the game of snooker.**

   Justification: Competition is a key part of snooker, with the winner scoring the most points in a frame. Users must be able gauge their progress in the game in order to know if they are currently winning or losing, else this competitive element is lost.

   Source: Gathered from MDA Framework.
3.8. Requirements Validation and Conclusion

Extensive requirements validation at this stage, such as prototyping, was not feasible in a project of this scale with the afforded timescale. The requirements were therefore reviewed by several potential stakeholders to confirm the features considered to be of most importance to the game were represented in the Requirements Specification. The only issue found was that several stakeholders thought that functional requirement 11, ‘The system should provide a method to control the angle at which the cue strikes the cue ball’, should be considered a high priority requirement, rather than just an optional inclusion. When pressed as to what feature would need to be compromised to ensure development time was afforded to this feature, they were uncertain as to whether any of the high priority features should be compromised for this feature to be implemented.

It should be noted that the reason this requirement was initially defined as optional was due to nothing more than the time limitations placed on the project. The Stakeholder Analysis showed that the cue angle feature was a reasonably high priority, but so were other key features such as power and spin effects. These features were assigned a higher priority due to them adding more important functionality to the game. It was therefore deemed necessary that cue angle adjustments should only be implemented if development time allowed.

After consideration of the alternatives, the stakeholders agreed that this requirement should remain as optional owing to the opinion that, although it is a feature that would add depth to the game, it does not add as much to the interaction experience as other high priority requirements.

The Requirements Specification has stated what features should be present in an interactive snooker game, and what should be achieved from the project. The next chapter looks at designing the system in respect to these specified requirements.
Chapter 4

Design

This chapter will divide the design of the system into four high level design areas. These areas are:

- Graphics
- Game Physics
- Rules
- User Interface

Each of these areas will be looked at in more detail, evaluating the key design issues that each section poses. The decisions made will subsequently be documented and justified.

4.1. Prototyping

Prototyping is an important stage of the design process for any system, as it is a technique that enables stakeholders to explore the imagined uses of the system being created. The significance of prototyping is apparent in a project of this nature, where the system is built around the graphical representation and interactivity of features.

Eric Todd, a senior game developer at Maxis, discusses the problems of designing without the use of prototyping (30), stating “words are fundamentally a terrible way of communicating interactivity”. Preece et Al (12) also states its importance; “A prototype allows stakeholders to interact with an envisioned product…and to explore imagined uses”. Stakeholder feedback during the design process ensures that the game is shaped into a product that corresponds with how the user would like to interact with the game.

As stated in the Requirements Analysis, it will not be feasible to produce a high-level prototype within the timescale of this project. However, low fidelity prototyping, in the form of paper-based screen mock-ups will be used
to discover if the design conforms to what stakeholders would like to see featured in the implementation.

4.2. Design Principles

There are a number of design principles that are promoted throughout many texts and papers. We will look briefly at the design principles outlined by Dix et Al (31). These are divided into three categories:

- **Learnability** – How easy it is for novice users to interact effectively with the system.
- **Flexibility** – The variety of ways in which the user can exchange information with the system.
- **Robustness** – Features that support the successful achievement and assessment of goals.

The most important of these principles in relation to this project are Predictability (Learnability) and Task conformance (Robustness).

Predictability ensures that any user knowledge of their interaction history is enough for them to be able to predict the result of future interactions with the system. This is important in our game, as users expect a past interaction to produce similar results when the interaction is reproduced. If an interaction produces differing results, the user will inevitably become frustrated. Task conformance is the degree to which the system supports all of the tasks of interest and whether it supports these as the user would like. A user will have set ways in which they will want to achieve the system goals; i.e. in snooker this could be potting a ball. We must ensure the design considers the most popular tasks, designing them in a way that supports users’ desires.

Further principles of relevance to this project are described in Appendix D (section 1). We must consider these principles throughout the design process and how they specifically relate to this project.

4.3. Graphical Design

As stated by Rabin (2), ‘Graphic design is a communication process through which ideas and concepts are presented in a visual medium. Using graphics elements, composition and colour, information can be presented in a very powerful and compelling way. Bad design, on the other hand, can interfere with the presentation of information. In game development, this translates to user frustration and negative game experiences.’

Rabin goes on to state five graphic design principles; Simplicity, Consistency, Know the target user, Colour Usage and Feedback Mechanisms. Simplicity, consistency and colour usage will not be described here, although these principles are outlined in Appendix D (section 2).
Know the target user relates to understanding how the product will be used by potential stakeholders. Consideration must be given to how knowledgeable the user is and how they perceive the information presented (2). To ensure a firm understanding of what stakeholders will want from the system, they will be included throughout the design process, asking them for their opinions on potential ideas and prototypes. Feedback Mechanisms are visual and auditory elements that help the user understand their interaction with the user interface, sometimes telling the user that they have accomplished a task. This can include implementing a visual highlight change or the appearance of a visual prompt.

The importance of good graphical design is apparent, as this is the first point of contact with the user. We will now look at how this can be best achieved in our project, adhering to the design principles described in Section 4.2.

4.3.1. Graphically Representing the Game

As previously discussed in the requirements, the graphics should give a realistic representation of the game of snooker. Therefore, graphically, the main design decision is to determine those elements of snooker that must be visually recreated, to ensure that the user immediately recognises that the game being represented is snooker. We must then consider how these elements can be translated into the virtual world.

Snooker has three key physical components that must be graphically implemented in the system; the snooker table, the balls and the cue.

The table needs to be recreated relative to the dimensions specified in the WPBSA rules (29), recreating the main body of the table, the cushions and the pockets accurately. These aspects of the table are the only areas of the table that affect how snooker is played; the table’s main body and cushions constrain the balls’ movement and the pockets allow for the successful potting of the balls.

It is not a necessary requirement to represent aspects of the table that only add to the game aesthetics, such as the table legs. Instead, the table can be visually represented as ‘floating’ in the virtual world. When interviewing stakeholders on this proposal, it was of little concern to them, with the majority not putting very much value on the graphical depth of the game. Many said that as long as the main features of the table were present and rendered realistically, then the interaction with the system was of a much greater priority.

In a typical game of snooker there are 22 balls on the table at the start of the game. This must be reflected in the system, with 15 red balls, 6 coloured balls (yellow, green, brown, blue, pink and black) and a white cue ball. The distinctive graphical feature of a snooker ball is that it is spherical. If merely ambient lighting was applied to each ball they would appear flat, yet the balls should exhibit a sense of volume to give the realistic impression of them rolling on the table. In order to exhibit this in the system, and to satisfy functional requirement 1.3, diffuse and specular lighting will need to be applied to each ball.
The cue is not complicated in terms of its design and so we will not consider this in too much detail here. The basic representation should be that it appears behind the ball showing the direction in which the ball is aimed.

There are several more advanced visual effects that should be considered, but will depend largely on the time required to implement them. Shadow effects will be necessary in the game to show definition where the cushions meet the table base, so that users can clearly see the table boundaries. Shadows could be developed further by adding shadows under the balls on the table. This is an effect that is an aesthetic addition to the game and so if implementation of this is discovered to be relatively complex it should be treated as a much lower priority to the more important game features.

Texture mapping is another effect that would add to the realism of the game, essentially allowing the attachment of images to polygons to provide more realistic graphics. An example of how this could be used would be the application of a wood effect texture to the base of the snooker table. The visual effect produced through texture mapping could not be achieved using basic material techniques. The decision to implement this will essentially depend on how much development time would be needed to dedicate to it, similar to the shadow effect discussed above.

In real life, several other physical entities are present that would be considered for implementation in a larger scale project of this nature. This was illustrated in the Evaluation of Existing Systems (see section 3.2) by the graphical depth of World Snooker Championship 2005. These entities can include virtually representing the players, referee, rests and the arena in which the game is taking place. We will not aim to virtually represent these features, as the focus of this project is on providing a realistic game experience, as well as providing realistic graphics. It would therefore be of little value to consider allocating development time for features that simply add graphical depth, over features that provide additional functionality.

We must also consider the implications of the users’ system setup on the game, to meet non-functional requirement 1. There must be some consideration of the graphical resolution being utilised by a user. The system should be developed with some flexibility for a range of screen resolutions, allowing the user to specify their preferred resolution upon the game starting up. However, there is no apparent need for every possible resolution to be available as an option. We should develop the system to incorporate the most commonly used screen resolution; 1024x768 (32). For those users who utilise a resolution less than, or greater than this, the system will also provide a full screen option, which will resize the window to fill the users screen. Perhaps in future developments the system could be adapted to afford a wider range of resolution options, but for this project these two options will be sufficient.

4.3.2. Visual Interaction

The visual interaction with the game controls not only affect how the user can move around the game world, but can also be manipulated to affect the difficulty of the game. A decision regarding how many camera views are offered to the user must be made, as well as ensuring that the views
implemented are of assistance to the user and are not simply included to increase the number of features available.

We must consider the provision of camera views that accurately relate to how a player would interact when playing snooker in real-life. Translating this interactivity to the project has been discussed in both the Literature Survey and the Requirements Analysis. In snooker, a player can walk around the table viewing the table from any angle before deciding on the shot to play. The standard method of representing player movement in virtual representations of snooker is to place the camera behind the ball, allowing rotation around the cue ball to select the direction of aim, and in some games, vertical movement to view the table from a greater height, as shown in Figure 4.1. This can then be supplemented with viewpoints that the user can navigate between, such as an overhead view of the table, to aid their choice of shot.

A more realistic interaction could potentially be provided by allowing the user to manually move the camera around the table to look at the table from any position and angle. The camera would then need to return to the location of the cue ball for the player to take their shot.

Sketches of these camera views (see Figure 4.1) were reviewed by several stakeholders to get their opinions. The majority expressed that they thought that manually viewing the table from any position would be a positive feature, but those who claimed to have previous experience with snooker games said that it would not necessarily be a feature they would utilise often, as the default view gave them enough information in the majority of situations.

Stakeholders were also asked to consider the possible benefits of supplementary viewpoints, and were asked for their opinion on a number of prototype viewpoints (see Figure 4.2 for examples). All those questioned said that supplementary viewpoints would be beneficial to the interactivity of the game and help them with their choice of shot. The overhead view was the most popular with stakeholders, followed by the option to zoom in/out on the cue ball. None of the other viewpoints were considered useful features that users’ would frequently utilise when playing the game.

Figure 4.1. The default camera system (left) and the manual camera view (right)
There is one major design issue when allowing the user to manually position the camera; how can we design the control system to allow for movement along both horizontal axes and rotation around the vertical axis? Mouse input would be the most favourable technique for this, but is much more difficult to implement in comparison to the favoured method of keyboard input. However, we have found no way to design the controls, in relation to keyboard input, to allow for this movement without overcomplicating the control system for users. This means that although this interaction would be of benefit, it is an unrealistic goal for this iteration of the game and the system should not be designed with this feature in place.

The aim will therefore be to implement a default camera view that enables users to rotate horizontally and vertically around the cue ball, as well as being able to zoom the camera in and out on the cue ball. An overhead view of the table will also be accessible to the user. Free movement around the table will not be a feature we aim to implement in our system, but could be a possibility for future versions of the game.

### 4.4. Game Physics

To ensure the game physics are realistic, meeting functional requirement 2, there are some real world constraints that must be satisfied. Balls must react like solid objects and so must collide when they hit one another. The specific physical implementation of ball physics is discussed in the Literature Survey.

Another important feature of the game is for a ball to fall down into the table pockets realistically. For this to occur, the system must take the affect of gravity on the balls into account. When balls then move off the solid table surface, as they would when they enter a pocket, they will naturally fall down into the pocket hole, as required.

The final factor that should be considered is how the balls realistically move around the table. In real life, the balls naturally slow down before eventually coming to a standstill due to the friction acting on them. This is a feature that must be recreated in the system to accurately mirror the real game.

Collisions, gravity and friction are the three main physics that must be implemented in the system. How each will be specifically implemented is an implementation decision and will be discussed in Chapter 5.
4.5. Rule based Design

The requirements specification details what rules must be applied to give an accurate representation of snooker. There are only a few design issues when considering how the rules will be employed in the system. These are what the system will do with balls that have been potted and are not to be re-spotted and the way in which the nomination of coloured balls should be handled in the game.

For the purposes of this system, a ball will no longer be visible once it has been potted and if it is not to be re-spotted to the table. Both the graphical and physical representation of the ball should be removed from the system. If only the graphical representation of the ball was to be made transparent, the physical entity would remain, which could cause interference as the game progresses and more balls are potted; i.e. the physical bodies of the potted balls could block the pocket holes. It is therefore necessary to also remove the physical representation of potted balls from the game.

How users nominate the colour they wish to aim for is a difficult design consideration. In more sophisticated commercial games, the colour is nominated by selecting the colour onscreen, often by clicking on it with the mouse. This is an unrealistic goal for our system, as we need to keep implementation achievable within the timescale of the project and as stated before, implementing mouse interaction can be a difficult process. There are two alternative approaches we could utilise. The first would be to allow the user to specify the nominated colour through keyboard input, perhaps by pressing a number in the range 2-7 (representing yellow to black). The second approach would be to allow the user to nominate by simply aiming the cue ball at their chosen colour.

Both of these approaches are feasible for use in this project. However, the first would require more output from the system, asking the user to input their nomination choice and displaying the resultant input graphically onscreen, as well as complicating the control interface with further key inputs. The second approach would not require this level of communication with the user, as well as eliminating the possibility of the user making a mistake when inputting their chosen nomination. This view was also supported when reviewed by stakeholders, many of whom commented that they did not want the hassle of inputting a number whenever a red was potted, but should be able to simply aim at the colour they wish to pot.

The second approach will therefore be the method utilised for nomination. Although the preferred approach, some problems do still exist which must be considered more carefully in the implementation of the system. These include what happens when a nominated ball being aimed for by the user is not actually hit, as this will adversely affect the score calculations, as the system will not recognise this as a foul.
4.6. User Interface Design

4.6.1. User Interface Design Guidelines

The system must conform to some standard principles of user interface design. Shneiderman defines a number of user interface design principles (1). The two principles of most importance to this project are Consistency and Minimal Surprise

- **Consistency** relates to the idea that system commands and controls should be kept constant in order to reduce learning time and avoid user confusion. It should be possible for the user to perform a task in exactly the same way over and over again. If this is not possible, users cannot repeat tasks easily and cannot gain a familiarity with the system controls. This in turn will frustrate users and may cause them to abandon the system altogether.

- **Minimal Surprise** concerns the idea that the system should not behave in an unexpected way. If an action in one context causes one type of change, it is reasonable to expect the same action in a different context will cause a comparable change (1). If the system behaves unusually, the user will be both surprised and confused.

The systems user interface must be designed with Shneiderman’s principles in mind, to ensure the system attains a high standard of usability.

4.6.2. Output considerations

Output from the system, with the exception of sound effects, will be presented in the form of graphical prompts or displays. We need to consider the most effective ways to display these outputs.

**Score display**

The most important output in relation to the gameplay is the score of each player in the game. The scores need to be shown clearly onscreen, in accordance with functional requirement 5, to ensure users are not confused by the output, instead realising its relevance to the game being played. The method in which the scores are displayed must be visually succinct and should not interfere with the user’s view of the game itself. The best approach to succeed in achieving these requirements will be to split the game screen into two frames (see Figure 4.3). The main frame will show the features of the game itself, such as the game table, the balls and the state of play. A secondary frame will then be displayed at the bottom of the screen and will be the location of the player scores, also displaying the player’s current break in brackets.

This design was reviewed by stakeholders who were of the opinion that this was a fairly natural way to display the players’ scores. The only concern was the clarity of the function each element in the display performed. The display should therefore explicitly label elements of the display, such as a player’s break score. It would also be beneficial to provide an explanation of the
graphical user interface (GUI) in the system user manual, should one be produced.

This two-frame structure is the visual backbone to the system and all other elements of the GUI must be designed with this in mind.

![Diagram of the two-frame interface structure]

Figure 4.3. The two-frame interface structure

In addition, to show users whose turn it is to take a shot, the player at the table will have their name highlighted, while the opponent’s name will be shaded grey. The convention of using colour to show features that are currently being used, whilst those that are not appear more washed out or shaded, is not a new concept. Shading, as discussed by Preece et al (12), has been utilised for this purpose in GUIs for many years. Common examples include many Microsoft applications’ menu interfaces where shading differentiates the currently available options from the restricted options.

**Shot information displays**

The aspects of the user interface we should consider next are the elements which relay information on the shot being played. This is effectively relaying back information the user has input into the system. These elements include the power indicator, the spin indicator, vertical cue movement and nomination.

The power indicator could be handled in one of two ways. The power could be represented by a graphical power bar which appears to fill up as more power is applied. The alternative to this is to represent the power as a percentage indicator, from 0-100%. For either method, the user would be able to increase or decrease the power shown by pressing the relevant controls on the keyboard.
Figure 4.4. Power indicators

Before the two possibilities were shown as prototypes to stakeholders, they were asked how they would expect the power to be represented in the game. The power bar was the only suggested method at this stage. This can be attributed to the fact that many similar types of game represent the power indicator in this way. Interestingly, when the two possibilities of power indicator were shown to stakeholders as low-level prototypes (see Figure 4.4), there was no clearly favoured method. A number of those reviewed said they preferred the fact that the percentage indicator put a clear measure on the power being applied. As no method proved to be more popular than the other with the stakeholders, the percentage indicator will be implemented as it can be most readily fitted into the aesthetics of the systems two-frame structure, whereas the power bar would interfere slightly with the main view of the game.

The spin indicator is much less of a design issue than the power indicator, in terms of how it can be represented. Through research and discussions with stakeholders it has become apparent that the best solution is to implement the spin indicator as shown in Figure 4.5. The visual representation conveys what spin is being applied to the ball on each shot by showing where the cue will hit the cue ball. This was also the case for conveying the vertical cue movement of the shot, if implemented. An indicator showing the angle of the cue would be present, as shown in Figure 4.5.

Figure 4.5. Spin and vertical cue alignment indicators
Aiming system

The Requirements Analysis identified the design of the aiming system as an aspect of the system that must be considered carefully. This is mainly due to the affect it can have on the overall difficulty of the game itself and the challenges posed of the user. A poor aiming system will greatly affect the way the user can interact with the game and could decide whether the user will want to keep playing the game and learn how to improve, or simply give up.

We asked potential stakeholders what their opinions were on how the aiming system should be designed. The majority stated that their main priority was that the aiming system was accurate and gave a clear impression of the direction the cue ball would travel in. This could be in any form as long as the aiming system was immediately clear to them without any ambiguity.

An aiming system similar to that used in World Snooker Championship 2005 could be employed, whereby the direction of the cue ball is specified by showing the potential path of the ball being targeted. The main disadvantage to this system would be that it would need refining for each ball in play dependent on its distance from the pocket. Simply using one set length of aim line would mean that for some balls the aiming system would explicitly tell the user where to aim in order to successfully pot the ball, as it would draw the line right up to the pocket. Evidently, this would take away any sense of challenge. Fine-tuning required to ensure this would not occur, would require development time that we cannot allocate due to the development needs of other aspects of the system.

A sensible alternative, which we will utilise in this project, is to show the potential direction of the cue ball by drawing a line through the centre of the ball, as utilised in FooBilliard. This ensures that users can clearly see the direction the cue ball will travel in, whilst providing a system in which users can hone their skills. The more frequently they play the game, the better they will get at judging the angle of each shot with respect to the aim line, conforming to user requirement 2.

Event Triggered Visuals

When the end of a game is reached, the system should make it clear which player has won. It will do this by displaying an output across the main game screen informing the user which player has won the game, writing simply ‘Player x has Won!’. This can be displayed prominently onscreen as the game is over and so the output can no longer interfere with the gameplay. The use of colour for screen output must be compliant with the colour scheme used for the rest of the user interface to ensure that the system conforms to Shneiderman’s UI principles (1).

Help screens

Our system will not include an extensive help system, due to the focus of the project being placed on the realistic implementation of the simulation, but it is necessary to implement a help screen briefly detailing the control system and
rules of the game, as stated by functional requirement 13. This would provide any stakeholder with the tools to use the system without any prior experience of the system, or of the game of snooker, fulfilling user requirement 1.

The most important characteristic of any help system is that it is easy to use, otherwise providing help can become a recursive process, as stated by Covin and Ackerman (33), ‘how can you effectively use help when you do not know how to use the help system?’ The help screen should therefore be easily accessible and communicate its information clearly to the user. This information should include a brief summary of the game rules and an explanation of the game controls.

**Sound effects**

Sound effects are not a necessary requirement of the system, but should be considered briefly in the design, should development time allow for their implementation. According to Steve Rabin (2), the two key points to audio design are creativity and integration. Creativity will not be discussed for the purposes of this project, but the integration of sound into the game world is more relevant. Should audio be integrated into the game, only simple effects will be present, such as the balls hitting one another. The most practical way of attaining these effects would be to digitally record them. The audio effect will then be triggered by two balls making contact with one another. How this sound effect is implemented into the system would also need to be investigated further upon its integration into the game.

**4.6.3. Input considerations**

The usability of any game relies heavily on the ease with which users’ can interact with the game through the control system, a point that was emphasised during the discussion of game interfaces in the Literature Survey (see section 2.2.3.1.). The design of the game’s control system should therefore be considered carefully to ensure that all features within the system are easily accessible by the user.

The decision was made to design the control system with respect to keyboard input only. As the number of features included in the system will be relatively low when compared to those of a commercial game equivalent, it was decided to keep the controls uncomplicated, with all operating through one piece of hardware, rather than the inclusion of further input devices, such as a mouse. User opinions were considered before this decision was finalised. When questioned regarding how they would like to see the control system implemented, none expressed concern over the decision to solely utilise the keyboard, a number stating that they liked the prospect of the controls being compact and uncomplicated by mouse controls. A few individuals did state that they preferred utilising the mouse for power control, but would not be put off by the system utilising an alternative if the control system was still accessible.

To comply with usability requirement 1.1, we must create a logical control interface. This will be achieved through applying a common keyboard
interface convention, mnemonics. A mnemonic is defined as a memory aid, employed in the context of keyboard input to help users remember the appropriate input to perform system actions. This will be utilised in the framework of our game by representing actions in this way. For example, to set the speed of the ball the user will hold the ‘S’ key while using the arrows keys to increase and decrease the speed of the shot. In games, it is also common to utilise the more distinctive keys on the keyboard to provide input. These keys include the Spacebar, Enter, Shift, Tab, Control and ALT keys. The relative size and position of these keys provide memorable input controls. The control system should make use of both these conventions, making certain that their application is sensible, locating the primary input keys within a reasonable distance of one another and not sparsely distributing controls over the keyboard.

A full list of controls cannot be finalised until it is known exactly what features have been implemented in the final system, but we will formally specify the controls of the potential features outlined by the Requirements Specification. Any modifications to these controls will be detailed in the implementation.

| Left Arrow key | Rotate Camera Right |
| Right Arrow key | Rotate Camera Left |
| Up Arrow key | Zoom Camera In |
| Down Arrow key | Zoom Camera Out |
| Spacebar | Take Shot |
| Tab | Switch to Overhead Camera View |
| Shift + Left/Right Arrow key | Fine Tune Rotation |
| A + Up/Down Arrow key | Control Vertical Movement |
| S + Left Arrow key | Decrease Speed of Shot |
| S + Right Arrow key | Increase Speed of Shot |
| M + Arrow key | Move cueball (when available) |
| E + Arrow key | Add Spin to the Cueball |
| B + Up/Down Arrow key | Raise/Lower Cue Butt |
| H | Help Screen |

**Figure 4.6. The Game’s Keyboard Controls**

### 4.7. Design Conclusion

The design of the system has now been detailed in full, describing how we aim to achieve the requirements specified in section 3.7. Stakeholders were involved throughout the design, refining initial prototypes where necessary.

It is important to note that not all of the requirements have been considered at this stage. Some of the requirements are, without doubt, implementation considerations. Usability requirement 6 (relating to the frame rate of the game) will depend on the API chosen and it would not be wise to make a decision on this until we have a better understanding of the implementation issues.
Functional requirement 3, which specifies the rules to be satisfied, is also an implementation consideration, as many of the rules are explicitly specified and do not therefore need to be considered during the design stage. We will, however, need to consider how these rules can be implemented into the structure of our system.

The next chapter details how the design solutions described in this section can be applied in the implementation of the system, subsequently describing the key details of the implementation of the game.
Chapter 5

Implementation

This chapter details the stages of system implementation, documenting important implementation decisions, as well as explaining the methods utilised. The design solutions discussed in Chapter 4 will be implemented into the system.

5.1. Implementation Language and API

In the Literature Survey, we discussed the need to utilise a graphics API and a physics engine in the implementation of the system. The choice was narrowed down to the two APIs which at present are considered the current industry standard; OpenGL and Microsoft’s Direct3D. From our research, as detailed in Section 2.3., the graphics API that most suited our requirements was the OpenGL API and this will be the API used in the development of the system.

The choice of physics engine was more clear-cut, owing to the relative lack of appropriate freeware solutions available. The Open Dynamics Engine (ODE) was found to provide the best solution, the details as to why are likewise specified in the Literature Survey (see Section 2.4.).

We must now decide on what language will be used for the implementation of the project. It is necessary to utilise a language that the developer is familiar with, as they must already learn to develop using OpenGL and ODE, neither of which they have previously utilised. The language must be well documented with good support and should support an Object-Oriented style of programming, as this is the style in which the developer is most familiar and will look to employ.

The language that will be used, and so meets these requirements, is C++. Java was strongly considered and supports many of the systems language requirements, but the ODE physics engine features a C++ API, with no comparable Java support available. This should ensure relatively easy integration with the rest of the system. C++ can also utilise the Visual Studio
IDE, which is accessible to the developer and provides excellent debugging facilities. C++ is an industry standard in professional game development due to its speed and power, with the majority of current games being developed in C++, exhibiting its potential. Although not required for the scope of this project, C++ provides cross platform support, which compliments OpenGL’s similar functionality. This could be exploited in future extensions of this project.

In summary, the project will be developed using the OpenGL 3D graphics API, the ODE physics engine and Visual C++. These provide the most suitable solutions for successful implementation of all aspects of our system.

### 5.2. Pre-Implementation Work

Two weeks were taken at the start of the implementation process for the developer to become familiar with the OpenGL API, developing a number of programs exhibiting basic OpenGL functionality. This was an important process to ensure that when it came to implement the project itself, the implementation was not delayed frequently by the developer not knowing the basic structure of the API.

This was similarly the case when the project reached a stage where the implementation of the game physics was possible. A few days were taken to look through the ODE documentation, as well as discussing the basic processes involved with Florian Schander, a post-graduate at the University. This enabled the functionality to be assessed and incorporated into the game, with a relatively minimal amount of problems encountered.

### 5.3. Code Architecture

The coding structure of the system has been influenced strongly by Astle and Hawkins (22), separating the operating system-specific code (the winmain.cpp class) from the OpenGL application-specific code (the CGfxOpenGL.cpp class). This allows the application to be readily portable, by rewriting the winmain class for other operating systems, a characteristic that could be taken advantage of in future developments. For the purposes of this project, the project will be written for use with Microsoft Windows XP.

The project also utilises two further classes; TableModel.cpp holds the code that draws each element of the table model and CCamera.cpp allows for the creation of a camera object that can be positioned in the game world by calling its accompanying functions. These functions are discussed in more detail in section 5.2.1.

Code has been commented throughout implementation to ensure that anyone with a basic understanding of OpenGL, C++ and ODE could look through the system and obtain a general understanding of the processes that are taking place at each stage.
5.4. System Implementation

The system implementation will be divided into four sections, as the system design was in Chapter 4. These four sections reflect the natural order of implementation for this project:

- Visual Implementation
- Physics Implementation
- Rule Implementation
- User Interface Implementation

Separating the implementation process in this way enables us to focus on the specific decisions made at each stage of system development, whilst also replicating the flow in which the implementation process was undertaken.

5.4.1. Visual Implementation

Modelling the Snooker Table

The system design identified that the most important elements to be represented graphically in the system were the snooker table and the snooker balls. During the pre-implementation stage, it became clear that OpenGL included functionality for drawing basic geometric shapes, such as spheres. Therefore visually portraying the balls would be fundamental. More complex was representing the snooker table in OpenGL, due to its more complex geometric shape.

Although the main structure of the table could be made up of rectangular primitives, features such as the cushion knuckles and pocket holes in the table were impractical to depict using OpenGL primitives, as specifying the location of hundreds of vertices was simply not a feasible implementation process. The practical solution to this problem was to utilise a 3D modelling package. This has enabled the development of a much more accurate model that could then either have been imported directly into the game world, or converted into OpenGL code in the form of a vertex array or a display list.

A number of freeware 3D modelling packages were considered. The main issue with the majority found were that they didn’t provide functionality for outlining the pocket holes in the snooker table, but would have required specifying each individual vertex location. This process would hold no benefits over simply specifying them directly in OpenGL.

The package found to be most suitable for our requirements was 3D Studio MAX 8 (3DS), for which a 30 day trial version could be obtained. Although only a trial version, it featured all the functionality we required for modelling the table. The main advantage to 3DS is that it is an industry standard piece of software, used in projects requiring far higher levels of visual detail. This meant that it was more than capable of handling the modelling requirements of our system; Pocket holes could be modelled by simply using a Boolean operation between two primitive geometric shapes.
The subsidiary issue was then how the model could be transferred into the game world. OpenGL.org (34) suggests that those developing using 3DS should investigate an export format called ASE. From further research into this format, a program called ASE2Prog developed by Philippe Kunzle, was discovered and utilised. ASE2Prog enabled the conversion of a .ASE file into an OpenGL vertex array (indexed or non-indexed) or display list.

For better clarity in the code structure of the project, the model was divided into numerous sections before conversion, subsequently being placed into separate functions within the TableModel class. This has ensured that different lighting and colour effects could be applied to each section of the table; the cushions, table base and pockets. This is necessary as, for instance, the cushions are required to have a material colour value of green, whilst the wooden base should have a material colour value representing brown, to mirror the look of a real snooker table.

**Advanced Lighting**

As specified in the design, advanced lighting was to be applied to the balls in the game to ensure the balls appeared realistic and not simply as flat entities (functional requirement 1.3). This has been achieved by utilising the material properties provided by OpenGL. This allows each ball to be defined by a number of related material properties, including its shininess and the colour of its ambient, diffuse and specular components, without the need to know any of the underlying mathematics involved in the production of the resultant lighting effects.

One problem emerged when implementing the specular component of each ball. On initial implementation, the balls were displaying unusually large specular highlights; in extreme cases these encompassed the entire ball. When this was investigated, it became evident that this was due to the setup of the game world. The elements in our game world had been scaled down to make the table model a more appropriate size for viewing. The lighting had not been affected by this scaling and therefore the lighting effects were approximately one hundred times greater than what was expected. This scaling was removed, with the viewing transformation of the scene recalculated to incorporate this, resulting in the accurate depiction of specular highlights on each ball (see Figure 5.1).
Setting up the Camera

To simulate movement around the game world, a virtual camera had to be implemented, which could be controlled by the user. As outlined in the design, its view must be centred on the cue ball, and allow for user controlled zoom and horizontal and vertical rotation. The functions that achieve this movement are contained in the CCamera class, whose attributes are shown in Listing 5.1.

```cpp
CCamera::CCamera()
{
    xPos, yPos, zPos = 0;
    xView, yView, zView = 0;
    xUp, yUp, zUp = 0;
    camR = 0;
}
```

**Listing 5.1. The CCamera class**

The system initially creates a new camera object at the start of each game, specifying coordinates for its initial location, view direction and up vector. It is important that CCamera holds values referencing these coordinates, as these are then used to update the OpenGL viewport using the `gluLookAt` viewing transformation. This update is carried out by the `SetCamera` function contained in the CGfxOpenGL class.

The camera's movement is controlled by three functions contained within the CCamera class:

(a) **LiftCamera** controls the vertical movement of the camera and is the most basic of the three functions in terms of its implementation. Stakeholder feedback showed that the vertical rotation of the camera, proposed by the design, was the lowest priority camera control proposed. It was found that vertical rotation about two axes would be relatively complex to implement and so it was decided that a compromise should be found. LiftCamera therefore performs a translation along the y axis. The direction of vertical translation is passed to the function, resulting in the camera’s y position being incremented or decremented in respect to this direction.

(b) **RotateCamera** controls the rotation about the centre of the cue ball around the y axis. The implementation of this involves the use of two fundamental trigonometric equations to find the new camera position along each of the x and z axes (see Equation 5.1.), multiplying the radius of the rotation circle by the relevant trigonometric function (sin for the x-axis, cos for the z-axis). Each point is then subtracted from the relative view direction to specify the new location of the camera.

\[
x = radius \times \sin(\vartheta) \\
z = radius \times \cos(\vartheta)
\]

**Equation 5.1. Trigonometric equations used in RotateCamera function**
(c) **MoveCamera** controls the camera’s ‘zoom’ movement in or away from the cue ball. This movement is achieved by first calculating the look direction by subtracting the view direction from the camera position, normalising the resultant vector. This is then multiplied by the desired speed of movement, before being added to the current camera position to give the revised camera location.

A need to constrain this movement was required to keep the user within an appropriate distance of the cue ball, and conversely stop the user from moving too close to the cue ball. The horizontal distance of the camera from the cue ball has been computed using Pythagoras’ Theorem on the x and z axes (see Equation 5.2.), with the camera’s movement restricted when the user attempts to breach the predefined distance restrictions. Restrictions taking into account the possibility of the camera moving an inappropriate distance along the vertical y-axis have also been put in place.

Should the LiftCamera function be improved upon in future developments by utilising vertical rotation about the x and z axes, rather than the current translation, Pythagoras’ Theorem for 3D space could be utilised to calculate the distance with respect to all three axes. This is not possible under the current implementation, as the distance restriction could be breached when adjusting the vertical movement of the camera and a ‘deadlock’ on the zoom controls subsequently reached. Although this could be improved upon in future versions of the game, the mechanism in place provides a suitable solution to perform efficient camera movement around the game world.

\[ \sqrt{xLength^2 + zLength^2} \]

*Equation 5.2. Pythagoras’ Theorem*

The keyboard input required to control each of these camera movements is discussed in section 5.2.4.

### 5.4.2. Physics Implementation

**World initialisation**

Two key initialisation stages had to be performed before any physics geometry could be applied to the system. The ODE world first had to be initialised to allow for any physical entities to be created and applied to the game world. This is then moved forward by ‘stepping the world’; an operation performed by calling the ODE dWorldStep function. The length of the time-step will control the accuracy of the collisions in our system; the longer the time-step, the less accurate the collisions will become.

```c
world = dWorldCreate();
...
dWorldStep (world,0.05);
```

*Listing 5.2. The initialisation and stepping of the physical world*
Gravity plays an important part of a system simulating a real life activity. This was defined in the created world to ensure physical entities, such as the balls, did not simply float unrealistically. Gravity is of importance to the system when simulating a ball falling realistically down into a table pocket on the snooker table.

```c
dWorldSetGravity(world, 0, -50.0, 0);
```

**Listing 5.3. The initialisation of gravity in the game world**

Both of these controls must be tested in order to fine tune the physics displayed by the system.

**Collision Detection**

ODE utilises geometry objects, or Geoms, as the fundamental objects in its collision system. Geoms can be placeable or non-placeable. A Placeable geom has a position vector and a 3x3 rotation matrix that can be changed during the simulation, whereas non-placeable geoms do not have the capability of position change over time, and therefore represent static features in the environment. These are defined in ODE by the assignment of a body. A placeable geom is defined by associating it with a rigid body structure, whereas non-placeable geoms have no such association. In our system the snooker table will comprise of a number of non-placeable geoms, while each snooker ball will be represented by a placeable geom.

The implementation of the non-placeable geoms that make up our snooker table is illustrated by Figure 5.2. The physical entities here were drawn by retrieving the location of each Geom and adapting an ODE class, drawStuff, to graphically represent each primitive. This graphical representation was necessary during development to specify the location of each physical primitive in the game world. The non-placeable geoms are displayed in red, with rectangular primitives used for the table base, cushions and the backs of the pockets, while cylinders were used for the pocket knuckles.

![Figure 5.2. Physical representation of the table using non-placeable geoms](image)
As mentioned above, placeable geoms are always associated with a rigid body structure, therefore each ball object has been assigned a rigid body as well as its geom. Rigid bodies are affected by pre-defined world constraints such as gravity and have the capability to have forces applied to them. We have utilised this force application for controlling the power with which the cue ball is hit. The locations of these bodies were used to draw the snooker balls, using OpenGL primitives to draw a sphere at each balls relative position in the game world.

In ODE, contact joints specify the movement of a body once a collision has taken place. The collision detection mechanism utilised in our system was adapted from the sample collision code specified by the ODE documentation. A simplified version of the systems collision detection mechanism from the nearCallback function is outlined in Listing 5.4. At each world step this function tests what possible collisions are taking place in the world. If the dCollide function returns non-empty, then a contact joint is made between the objects taking part in the collision; b1 and b2. Each contact joint has a number of associated properties that can be adjusted to affect the collision response.

```c

    dCollide contact[N];
    n = dCollide (o1, o2, N,&contact[0].geom,sizeof(dContactGeom));

    if (n > 0) {
        for (i=0; i<n; i++) {
            dJointID c = dJointCreateContact (world,contactgroup,
                                                &contact[i]);

            dJointAttach (c,b1,b2);
        }
    }
...
    dSpaceCollide (space,0,&nearCallback);

Listing 5.4. The nearCallback function
```

Five possible collision scenarios can occur in our system:

(a) Collision between a ball and the table base,
(b) Collision between a ball and a cushion,
(c) Collision between a ball and a pocket knuckle,
(d) Collision between a ball and the back of a pocket (or stopper), and
(e) Collision between two balls.

Each potential collision entity required the specification of its associated contact joint properties. This ensures the collision response accurately replicates the physical response of a comparable collision in a real-world game of snooker. The cushions, pocket knuckles and balls all required further specification of a bounce parameter to guarantee scenarios (b), (c) and (e) were simulated accurately. The bounce parameter accepts an input value between 0 and 1, with 1 being maximum bounce and 0 no bounce. Cushions and pocket knuckles were assigned bounce parameters of 0.8 and 0.5 respectively. Although in real life these two elements of the table are made of the same material and would therefore provide a very similar reaction when struck by a ball, it was found that balls were unrealistically bouncing back out of the
pocket when striking the pocket knuckles in the game. The knuckles were therefore assigned a slightly lower bounce parameter than that assigned to the cushions. The balls bounce parameter was adjusted several times before deciding on a parameter of 0.5. The final bounce parameters of each entity will be fine tuned during system testing in relation to user feedback, to achieve our requirement of accurate ball physics (functional requirement 2).

The table base only required the trivial specification of its contact parameter, to ensure the snooker balls do not pass through the base. As the balls maintain constant contact with the table base, no further definition was required. Similarly the stoppers did not require a bounce parameter to be specified. Balls entering a pocket will strike the stopper and, due to the absence of bounce, fall down into the pocket with respect to the pre-defined gravity.

It is important to note that when the cue ball collides with another ball on the table, the ball hit first must be recorded for use in the systems scoring algorithm, so that it can identify that a legal shot has been played. This case is therefore tested for in the system and when the first ball collision is detected involving the cue ball, the ball struck is stored in a variable called ballHit, for reference later in the scoring algorithm.

**Applying real world physics to the game**

From implementing the collision detection and subsequently exploring the ODE documentation, it was discovered that ODE failed to feature the functionality to apply friction to a rolling body. As all of the moving bodies in our game world are spherical entities that roll along the base of the snooker table, this caused an immediate problem to the implementation of friction in the game world.

It was decided that the system was to make use of ODE’s velocity functionality to slow the balls’ movement over time. The movement of the ball has been measured at each world time-step by comparing its current location to its location at the previous time-step. The magnitude of this movement then determines the size of the damping applied to the balls current velocity, to give the effect of the ball slowing down with respect to friction. Once the movement of the ball has fallen below a specified level, a velocity of zero is applied to the ball, completely stopping its movement. The scale of damping is another aspect of the system that should be fine-tuned in relation to user response in the testing.

The functionality in ODE to apply a force to the cue ball when a shot is taken is trivial, but the force must be applied in respect to the cameras current location in the game world and is therefore more complicated. The system must consider where the camera is positioned along both the x-axis and z-axis, scaling the force applied to the x and z directions relative to these positions. The mechanism used to achieve this effect is shown in Listing 5.5. The section of most importance is contained in the final ‘else’ statement. This calculates the relative x and z lengths as a percentage over the total length, xLength + zLength. These percentages are then used to scale the force applied in the x and z directions, applying the forces to the cue ball by utilising the ODE function dBodyAddForce.
Listing 5.5. The applyForce function

Implementation of spin effects

An issue that emerged during the implementation process was the inclusion of spin effects applied to the cue ball. Although considered a high priority requirement (functional requirement 10) in the requirements specification, the implementation of spin effects was discovered to be unrealistic within the timescale of this project. The physics required to implement spin effects are of an entirely different nature to any of the other ball physics present in the system and would require a large amount of research and development time to be focussed on implementing this functionality. It was subsequently decided that this feature would have to be omitted from development. Spin effects, although a feature that we would have preferred not to have omitted from the game, only adds to the user’s repertoire of shot options. The time spent developing this feature could be spent developing aspects of the game that add
5.4.3. Rule Implementation

Rule implementation was done with respect to the system design to meet functional requirement 3. We will look closer at some of the most important aspects of the rules implementation.

Specifying ball locations

The game world was setup to replicate the location of the balls at the start of a real game of snooker, as outlined in the game rules (see Appendix C). The only problem faced was related to how close the red balls were positioned next to one another in their triangular pack. The position of each red was calculated with respect to the balls’ radius, positioning the reds as tightly together as possible in a triangular pattern. However, when the cue ball struck the pack of reds from the ‘break off’ shot, unexpected collision responses occurred, with balls passing directly through one another. The only explanation we have found for this behaviour is that the red balls were in such close proximity that a contact joint was created between the balls upon the world initialisation, so when one was moved further into an adjacent ball, ODE simply failed to recognise the collision and the balls just moved through one another. The solution found was to move the reds a small distance away from each other. This is not an ideal solution as the red balls need to touch in order to accurately simulate movement upon being struck by the cue ball at the ‘break-off’ shot. This compromise does detriment the simulation of how the balls would react in a real game ‘break-off’ shot, but a better solution could not be found during development.

Evaluating the world state

The entire rule structure depends on the system knowing exactly when a shot has been completed and the game world has returned to a stable state, ready for the next shot to be performed. Once a stable state has been determined, the rule checking mechanism can then be applied to the world, carrying out the necessary scoring algorithms to work out the conditions reached during the last shot. This stable state check is carried out by the function getBallState (see Listing 5.6). The function analyses the position of each ball when compared with its position in the last time-step. If these are found to be equal, then it is considered still. If all balls in play are found to be still, a stable state has been reached and rule checking may commence, before subsequently allowing play to continue.

Re-spot mechanism

Balls potted are discovered by examining their y position in the world; balls that have fallen down a pocket will have a y value less than zero. Adhering to the rules of snooker, if the ball potted is a red, it is permanently out of play.
Its related geom is therefore destroyed, removing it from the scene, as detailed in the system design (see Section 4.5). If the ball potted is a colour and there are one or more reds still in play, the colour is re-spotted to the table in its appropriate ‘spot’ location. It is important to note that if the spot location on the table is occupied by another ball, the rules state that it should instead take up the position of the highest value spot location available (i.e. black highest, yellow lowest). If no spot locations are free then it should be placed as close to its own spot location as possible, whilst staying between its spot location and the top cushion. The implementation of these exact rules has been slightly compromised by development time constraints. Balls that require re-spotting to an occupied spot location are not placed on the highest available spot location, but are instead placed as close to the spot location as is possible, with respect to the rules mentioned above. The accurate implementation of the re-spot rule could be achieved in future developments of the system.

int CGfxOpenGL::getBallState()  
{  
  //if returns 0, all balls have stopped  
  //if 1, some are still moving and damping should be reapplied  
  
  int ballsStill = 0;  
  const dReal * position;  
  
  for (int i = 0; i<22; i++)  
  {  
    if (offTable[i] != 1)  
    {  
      if (i == 0)  
      {  
        position = dBodyGetPosition(cueBall);  
      }  
      else if (i > 15)  
      {  
        position = dBodyGetPosition(colourBall[i-16]);  
      }  
      else  
      {  
        position = dBodyGetPosition(redBall[i-1]);  
      }  
    }  
  }  
  
  if ((ballPos[i][0] > position[0]  0.00005) &&  
      (ballPos[i][0] < position[0] + 0.00005) &&  
      (ballPos[i][2] == position[2]))  
  {  
    ballsStill++;  
  }  
  else  
  {  
    ballPos[i][0] = position[0];  
    ballPos[i][1] = position[1];  
    ballPos[i][2] = position[2];  
  }  
  
  if (ballsStill == ballsInPlay)  
  {  
    return 0;  
  }  
  else  
  {  
    return 1;  
  }  
}  

Listing 5.6. The getBallState function
Related to this, to meet functional requirement 4 the cue ball had to be capable of being moved about the D upon being respotted, or at the start of the game. This movement can be achieved by holding down the M button and specifying the direction of movement with the keyboard arrow keys. This part of the system has only been implemented to a very basic level, and does not allow for movement relative to where the camera is positioned in the game world. So holding the M and the left arrow keys will always move the cue ball along the positive x axis, irrespective of the camera. This could not be rectified during this iteration of the project, but this functionality should be a priority if future improvements were to be made.

Score Calculation Algorithm

The algorithm used for calculating the score can be visualised as a decision tree for simplicity (see Figure 5.3).

![Flow diagram of Score Calculation](image)

The path taken at each stage depends on the state of the game world. When a node is reached with no further branches, the system calculates the score or corresponding foul. The point score of the foul is dependent on the reason for the foul. For example, if the correct ball was not hit, the index of the ball hit
would be passed to the foulIndex function in the system, which would return
the number of points conceded.

5.4.4. User Interface Implementation

The design explicitly detailed how the user interface should be presented. We
will briefly look at how these designs were applied in the system.

Control system

The control system was implemented in direct correlation to the controls
outlined in the design (see Section 4.6.2.). As some features were not included
in the final game, these were omitted from the control architecture. The
implementation of the game controls was a fundamental part of implementation
and will not be detailed here.

Scoring display system and Power Indicator

The score display and power indicator have been implemented as specified in
the system design. Figure 5.4 shows these sections of the interface as they
have been implemented in the system. Text was rendered by adapting a code
sample from NeHe.gamedev.net, enabling the specification of font variables
such as font type and size, before sending the specific text to the glPrint
function for display (see Listing 5.7).

```
 glPrint("Player 1 - %d", playerOneScore);
```

*Listing 5.7. A call to glPrint to display score information*

Notice the use of colour highlights in the display to signify that it is Player 1’s
turn to play the next shot, with the use of grey to imply that it is not Player 2’s
turn. This display has been implemented as an overlay at the bottom of the
main game viewport, as outlined in section 4.6.2 of the system design.

```
| Player 1 - 0 (Break 0) | 50.00% Power | 0 - Player 2 |
```

*Figure 5.4. The score display and power indicator bar*

Aim system

The design specified that a single aim line would be implemented in our
system. The system has displayed the aim line passing through the centre of
the cue ball, as can be seen in Figure 5.5. This outlines the path the cue ball
will take when the shot is initiated by the user. An event triggered visual,
displaying the end of the game was not implemented due to time constraints,
with higher priority functionality taking precedence in the development of the
system. This would be relatively fundamental to implement and would be a
logical future improvement should the system be developed further.
Help Screen

The help screen was kept very simple, as we could not allocate a substantial amount of development time to this element of the system and so implementation of a more complex nature was not possible. The help therefore briefly details the basic aims of the game of snooker, before outlining the control system. This was done by utilising the code sample from NeHe.gamedev.net, as utilised for the score display interface and visual prompts previously discussed.

**Game Rules**

Three players take rotationally. A player should first aim to pot a red ball (worth 1 point), and if successful then try and pot one of the coloured balls on the table. The coloured balls are worth:

- Yellow = 2 points
- Green = 3 points
- Brown = 4 points
- Blue = 5 points
- Pink = 6 points
- Black = 7 points

This remains the target (potting a red then a colour) until all reds have been potted. When this occurs the coloured balls should be potted in the order: of the order of value, lowest to highest.

The player with the highest score after all balls are potted is the winner.

**Game Controls**

- Left Arrow = Move Aim Left
- Right Arrow = Move Aim Right
- Shift + Left Arrow = Slowly Move Aim Left
- Shift + Right Arrow = Slowly Move Aim Right
- Up Arrow = Zoom In on Target
- Down Arrow = Zoom Out on Target
- A = Up Arrow = Move Camera Up
- A = Down Arrow = Move Camera Down
- TAB = Overhead View
- Press H to return to the game

**Figure 5.6. The Help Screen**
5.5. Implementation Conclusion

This chapter discussed the techniques utilised to implement the key functionality of the system. We must now test and evaluate the system to find any erroneous functionality and to fine tune certain aspects of the system, in particular the algorithms and variables that control the game physics.
Chapter 6

System Testing and Evaluations

This chapter is dedicated to the testing of the system. Testing has been used to validate the functionality of the system and to refine the real world physics demonstrated by the game. Various types of user have been tested interacting with the system, in order to gauge how well the system meets the requirements detailed in the Requirements Specification.

6.1. Test Plan

The system testing was divided into three areas, (i) Defect Testing, (ii) Integration Testing and (iii) User Testing. These approaches were utilised throughout the implementation process and further tests carried out upon the systems completion. It was important that an iterative approach to testing was adopted during the implementation process, to ensure that new functionality not only operated in the desired manner, but also conformed to user expectations. Errors could then be isolated, and user requests handled, with greater ease than if the testing was carried out solely upon completion of the system.

Each stage of testing was comprised of a number of subsections. Defect testing was made up of Black-box and White-box testing. Black-box testing derives test cases from the Requirements Specification, evaluating the systems inputs and outputs without studying its underlying implementation. Conversely, White-box testing derives its test cases from the implementation, evaluating the functionality of specific functions and subroutines. Where possible, we have tested all paths available within a tested component although this was occasionally difficult due to the relative obscurity of some of the rule implementation tests, such as testing how the system would react when three or more balls are potted. Where this was the case the system was set up specifically for these test scenarios, rather than the correct positions in accordance with the rules, although for some scenarios this was still not possible. The debugging mechanism in Micro Visual Studio will be used during White-box testing to track key variables within the component being evaluated to ensure data is handled as expected.
Integration Testing has been carried out on a relatively small subset of the system, as many of the systems functions work independently of one another. The major system components have however been tested for integration, checking that the functions involved in these mechanisms interact with one another correctly. This includes test cases monitoring the values passed between functions and the accurate updating of objects in the system when required. Subsequently, Microsoft visual studio will again be used to track this integration between functions.

User evaluations have been utilised to attempt to quantify how usable the system is, evaluating the level of difficulty pertaining to the performance of fundamental tasks within the system. Two evaluation techniques were utilised; Cooperative evaluation and Think Aloud evaluation. The Cooperative evaluations were conducted with one test user playing against one evaluator, who also made notes relating to comments the user made regarding the usability of the system and its features. The user was asked to confirm their expected output from the system during interaction and to describe the actual output received to ensure the game complied with the usability principles and guidelines the system design took into account (see Section 4.2 and 4.6). During the Think-Aloud sessions, users explained their interaction with the system to an evaluator, who studied and recorded any interesting observations or opinions expressed by the user. The evaluator could verbally prompt the user for responses by asking questions relating to their thoughts behind the features in the system. This evaluation technique was utilised to help users express their thoughts relating to system features as they occur, rather than asking them to try to recall these opinions later, as can be required when conducting questionnaires or interviews.

This range of testing strategies was deemed sufficient for discovering any functionality or usability flaws in the system. The utilisation of unit testing was considered, but regarded as excessive for the requirements of this project; the aforementioned testing methodologies being considered more than adequate to discover any errors in the implementation. We will not detail the plan for every test and evaluation carried out as this is inconsequential to the value of this report, instead focusing on the results of the approaches applied.

6.2. Defect Testing

6.2.1. Black-Box Testing

Black Box testing was carried out on three levels; control system tests, physics tests and rule tests. As stated previously in the test plan, these areas were tested throughout the implementation process after functionality of a feature had been completed. Testing in this way helped discover many errors before further features were incorporated into the system. We will detail the specific test cases carried out upon completion of the systems implementation, focusing largely on the test cases that produced erroneous output, before explaining how each problem was solved or further investigated. A full record of the test cases employed can be found in Appendix F.
Table 6.1. Black-Box Testing - Control System Tests

<table>
<thead>
<tr>
<th>No.</th>
<th>Test</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Press spacebar once to play the shot, then press space an additional number of times</td>
<td>Plays the initial shot upon first press. All further presses have no effect</td>
<td>Output as expected</td>
</tr>
<tr>
<td>2.</td>
<td>At the start of a frame, or directly after the cue ball has been potted, hold M and press one of the arrow keys</td>
<td>Movement of the cue ball around the D should be controllable</td>
<td>Output as expected</td>
</tr>
<tr>
<td>3.</td>
<td>Use a combination of the camera controls to test the boundaries for deadlock</td>
<td>Camera should always be controllable, with no deadlock scenario reached through a combination of camera movements</td>
<td>Problem found with the combination of vertical movement and zoom.</td>
</tr>
</tbody>
</table>

Control tests 1 and 2 (Table 6.1) both returned output as expected. Test 3 discovered a problem when the vertical movement and zoom camera controls were used in combination. It was found that if the height was increased using the vertical movement control, when then zooming out, the camera was not restricted in the y direction and could be moved back towards infinity. This was resolved by adding a further criterion to the 'if' statement contained in the moveCamera function. This restricts the camera movement to movement within a number of units distance from the cue ball.

Table 6.2. Black-Box Testing - Physics Tests

<table>
<thead>
<tr>
<th>No.</th>
<th>Test</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Ball hits a cushion</td>
<td>The ball should rebound off the cushion back into the area of play at an angle relative to the angle of collision</td>
<td>Output as expected</td>
</tr>
<tr>
<td>2.</td>
<td>Ball hits another ball</td>
<td>The two balls should collide, with a relative directional force exerted on the ball struck.</td>
<td>When high levels of power are applied to the cue ball the collisions become less accurate.</td>
</tr>
<tr>
<td>3.</td>
<td>Potting a ball with a large amount of power applied</td>
<td>The ball should hit the back of the pocket and fall down into the pocket due to gravity</td>
<td>Output as expected</td>
</tr>
</tbody>
</table>

Of the physics tests carried out, only one successfully identified a defect in this area. Physics test 2 (Table 6.2) found that when a large percentage of power was applied to the cue ball, the collisions became less realistic. When investigated, it was found that this was because the amount of movement between the ODE world time-steps was much greater when the ball was given more power than if it had had a small power applied, and this was causing less accurate collision responses. Only a compromise to this problem
could be reached, which was to reduce the maximum power the user could apply to the cue ball on any shot. To fully solve this problem and produce near perfect ball responses, an addition to the collision functionality in ODE would need to be developed. This could utilise some of the collision techniques outlined in the Literature Survey (see Section 2.4) whereby when a collision is detected, the time-step is moved back to the exact point of collision. ODE does not handle collisions in this way, instead evaluating the collision at the time-step when the collision occurs, leading to erroneous collision responses, as discovered.

Table 6.3. Black-Box Testing – Rule Tests

<table>
<thead>
<tr>
<th>No.</th>
<th>Test</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Hitting or potting a legal coloured ball when all red balls have been potted. Colours must be considered legal in the order yellow, green, brown, blue, pink, black</td>
<td>Hitting or potting a colour in this order is considered a legal shot, incrementing the current players score if a legal pot is achieved</td>
<td>Problem found with yellow ball being re-spotted at this stage of the game</td>
</tr>
<tr>
<td>2.</td>
<td>Pot the cue ball after hitting another ball on the table</td>
<td>Should increment the opposing players score by the relevant amount and re-spot the cue ball in the D. Should adjust the interface to show that it is now the opposing players turn</td>
<td>Output as expected</td>
</tr>
<tr>
<td>3.</td>
<td>More than one coloured ball is potted in one shot, when an illegal ball is hit first</td>
<td>Should increment the opposing players score relative to the highest value coloured ball potted (yellow, green, brown = 4, blue=5, pink=6, black=7), re-spotting the potted colours accurately and adjusting the interface to show that it is now the opposing players turn</td>
<td>Not as expected, illegal ball hit first was always the foul value</td>
</tr>
</tbody>
</table>

Two further defects were discovered through the test cases applied to the rule implementation. Test case 1 found that when all the red balls had been potted and the yellow ball became the target ball, a successful pot always resulted in the yellow ball being re-spotted by the system. On investigation of the checkPotted function, there was no functionality to stop the colours being re-spotted once this stage of the game had been reached. A certain amount of restructuring of this function had to take place to accommodate the solution. This functionality was subsequently implemented and further tests resulted in the expected output, with the yellow ball no longer being re-spotted and its physical body being destroyed, so it was no longer visible in the game.

Test case 3 is a very uncommon scenario in a game of snooker, but when setup and tested in the system, a flaw in the calculation of the foul was recognised. When two colours were potted in the same shot, the foul was relative to the
first illegally hit ball. The correct functionality should be that a comparison of the foul values of the illegal ball being hit, and the value of the two colours potted, should be made. The highest value foul out of these two calculations should be the foul value awarded to the opposing player. The score algorithm shown in the implementation (see Figure 5.3) needed slight modification (see Figure 6.1).

It is important to note that test cases could only be applied to realistic rule occurrences. It would have been ideal to test scenarios where 3 or more balls were potted in one turn, but in reality this was simply infeasible, due to the inability to set these scenarios up in the game. In these cases the code has been thoroughly checked to minimise the risk of an error occurring in these circumstances.

The score calculation was also tested for defects by playing through a number of games, predicting the expected score output and recording the actual output displayed by the score display of the game. An example of this can be found outlined in Appendix F, but no erroneous output was observed.

6.2.2. White-Box Testing

Black-box testing was used to find many of the high level system errors. White-box testing was therefore designed to test the underlying structure of the key functions within the system. Of the functions tested (see Appendix F), only one produced any erroneous output, which can be attributed largely to the resolution of errors during the Black-box testing.

**Function: calculateScores**

- **Test:** Legal ball (yellow) potted when no red balls left in play
- **Expected:**
  - score = score relative to colour potted
  - foul = 0
  - foulState = 0
Actual: not as expected, counted as foul shot
Score = 0
foul = 4
foulState = 1

It was found that the variable, onColours, which represents whether there are any further red balls to be potted in the game, was not being set to the yellow ball once all the red balls were off the table. Following the flow of data through the function, the point at which this variable should have been updated was identified and the relevant code changes made. The path was tested again after this modification and the output of the system was as expected.

### 6.3. Integration Testing

Integration testing was utilised to ensure that the functions, when integrated, performed as expected. The functions that performed the most important system functionality were tested and a full description of the tests can be found in Appendix F. A significant, although rare, error was identified in the functions that comprised the friction mechanism in the game. The test relating to this is outlined below:

**Friction Mechanism**

**Test:** Hit any number of balls on the table, to test the friction mechanism. Test over a variety of shots and scenarios.

**Expected:** When the applyFriction Function is called, check that for each ball damping is applied (applyDamping function) and then from this that the velocityScale function is called and returns the relative scale of damping for that ball. Ensure that damping is applied until the ball is classified as no longer moving.

**Actual:** Occasionally found that balls were still moving by a very small degree, even though a velocity of 0 was being applied to the ball.

This error was quite serious, as when this occurred, the game would reach a state of deadlock due to the end of a player’s turn never being recognised. On investigation, the cause of the error was attributed as an error in the ODE functionality, as the velocity was correctly being set to zero, but the physical body of the ball was still moving by a minute amount. To nullify this error, code allowing for the recognition of this extremely slow moving body was created and integrated into the getBallState function. When a ball was then discovered in this state, the system would move the game on to the next player’s turn, allowing the next shot to be played and, most importantly, avoiding deadlock.
6.4. User Evaluations

As has been stated throughout this report, a game is ultimately judged on the opinions of those who play it. Evaluations were therefore carried out in an attempt to gain a better perspective as to what areas of the game were successfully designed and implemented. We can then quantify how the system meets the usability requirements set out in chapter 3.

From both sets of evaluations, the key opinion conveyed by the participants was that no matter how certain features were designed, different users would have their own opinion as to how the game should have incorporated certain features. A specific example would be how the aiming system was implemented in the system. There was mixed feedback from users relating to how they would like to see the game incorporate this feature. Some liked the clear display and immediately adapted to how the aiming system worked, while others were initially confused and said that they would have preferred a more visually ‘obvious’ system. The most common suggestion of improvement was to show a visible cue behind the cue ball in the direction of aim. It is worth noting that all users did adapt to the system and by the end of the test game their shot success rate was much higher than it was when they started. This successfully fulfils user requirement 2 of the Requirements Specification.

One aspect of the system that was addressed due to feedback concerns raised was the way fouls were displayed in the game. Several participants highlighted the need to be informed clearly when a foul had been committed. This was indeed not obviously apparent in the previous version of the system, with fouls being added on but no visual prompt shown to the user to inform them that a foul had been committed. It was decided after consulting with users, that when the system records a player committing a foul, the game should make it clear that a foul has been committed by outputting a visual prompt stating ‘FOUL SHOT’. This had to be displayed in relation to the design considerations specified previously, so that the interface was consistent and the prompt did not hinder the player’s interaction with the system. This output was therefore implemented to be unobtrusive to the game being played but clear enough for the user to know a foul had been committed.

Features that were looked upon by participants as the strongest aspects of the system included the following:

• The majority of users commented that the graphics were of a good level, represented the real game of snooker successfully (functional requirement 1)

• The control system was well received, with many users finding it easy to use, memorable and logical, fulfilling usability requirement 1. The main reasons given for these opinions were that the controls were within close proximity and conformed towards other similar control systems the users had previously experience with. Those who had no previous game experience liked the simplicity of the controls, stating that if more features were added to future versions of the game, the control architecture should still focus on simplicity to avoid the user having to remember a large number of keyboard inputs.

• Ball collisions and realism were regarded as a strong aspect of the game, with no participants complaining of unrealistic reactions. It is an
interesting observation that the Black-box testing highlighted high speed ball collisions as a problem, but the participants showed no concerns of this nature. Users were asked to play shots at high speeds to see if the system reacted as expected and still no negative feedback was received as to the way the balls reacted. The representation of friction was also commented on by several participants familiar with the real game of snooker, stating that it was a very accurate representation of the ball physics present in a real game (functional requirement 2).

There were some common suggestions for improvements to the system. The most frequent was the lack of spin effects in the game. Participants did however say that a lack of this feature did not detract from the game, but that it would add a lot in terms of the realism of shot options available in the game. A suggestion to implement just the basic topspin and backspin options was suggested by one participant. The exact way in which this could be incorporated into the game should be considered if improvements to this aspect of the game are considered in future iterations of the system.

It was also found that participants with extensive knowledge of the game of snooker could discover the specific rules that were not implemented in the game. One particular evaluation that took place saw the game end up the state whereby, in real-life, the ‘free ball’ rule would come into effect. This rule has not been implemented due to the complexity of the implementation, which would not have been a realistic goal within the timescale of this project. Future implementations should try to maximise the number of snooker rules present in the system, as a key target audience for the game are individuals with good knowledge of the game of snooker. These users would become more frustrated at these discrepancies than those users who are less familiar with the rules of snooker.

The overall feedback from evaluation participants was very positive. Many of the features suggested by the users were discussed at some stage of development, some being ruled out due to further analysis carried out and other features were simply not feasible for implementation within the constraints of the project. The usability of the features in place can be regarded as a success, with the control system garnering the most praise from participants.

The next, and final, chapter concludes on the successes and failures of the project, how the project met (or did not meet) the requirements and reflecting on the key decisions made throughout development. Logical future improvements to the game will also be explored and recommended.
Chapter 7

Conclusions

The aim of this project was to design and implement an interactive snooker game, implementing the basic functionality needed in a game of this kind. This aim was achieved, and we will now evaluate how the system specifically complied, or did not comply, with the requirements specified in Chapter 3 (Section 3.7).

The Requirements Specification outlined a number of key requirements that were a high priority for the system. Two functional requirements were compromised during the system development, due to the realisation that the implementation was not feasible within the time constraints on the project. Functional requirement 1.1 was only somewhat compromised, as the table did not include the visual representation of table markings. This was due to the implementation requiring OpenGL texture mapping techniques, which as specified during the system design, would have needed to be researched by the developer before this feature could have been integrated into the system. The time needed for this to be integrated was, unfortunately, not available, due to the development time being allocated to the implementation of more crucial system functionality.

Functional requirement 10, relating to the inclusion of spin effects, also had to be omitted from development, as detailed in the system implementation (see Section 5.2.2). The ball physics required to implement spin effects could have potentially occupied a number of weeks’ development, and it was decided that allocating development time to the inclusion of this feature could consequently detriment the functionality of other areas of the system. The development time was therefore allocated to other high priority features, such as the rule implementation to ensure the key game functionality was in place upon the projects completion.

The other high priority requirement compromised was functional requirement 3.4.2, which states that the system must be able to cope with other balls occupying a re-spot location. This requirement was in essence fulfilled, but does not accurately adhere to the rules (see Appendix C). Instead of searching for the highest value available spot on the table, the system places the ball as
close to its own spot as possible, whilst also ensuring it is positioned between its own spot and the top cushion. This rule was compromised due to the rule occurring only very rarely and so was not considered a major rule discrepancy. However, any future iterations of the system should address the functionality of this rule, accurately implementing this feature in relation to the rules (29). All other high priority functional requirements were satisfied by the system implemented. Of particular success were the achievement of functional requirements 2 and 5. Functional requirement 2 specified that the game must feature ball physics that realistically simulate how the balls would move in a real game of snooker. This requirement caused the largest implementation issue, as it was discovered that the ODE physics engine did not support rolling friction, which was specifically required in the game. Subsequently, a velocity damping mechanism had to be developed and integrated into the system. User evaluations identified that the realism of the game physics was regarded as one of the best aspects of the game, which suggests that the friction mechanism achieved its goal of emulating the real-world force.

Functional requirement 5 stated, ‘The system must provide an accurate scoring system’. Through extensive defect testing and evaluations the scoring mechanism was refined, so that it now accurately calculates all the implemented scoring conditions. In addition, throughout the evaluations conducted, no scenario was reached in which an incorrect score was displayed. From the testing and evaluations carried out, we can therefore conclude that the scoring system is near 100% accurate, with respect to the implemented rules, although it was infeasible to test every possible scenario due to the complexity of achieving all possible game states, i.e. potting three or more balls in one turn.

In relation to the usability and the user requirements specified, the majority were achieved by the system. During evaluation the control system received a great deal of positive feedback from participants. All participants needed no further guidance relating to the system controls than what was specified by the evaluator at the beginning of the evaluation, with none of the participants requiring the use of the help screen. This implies that the interface was immediately memorable to users, with the controls specifically designed for simplicity of use to enable this requirement to be achieved (usability requirement 1).

The competitive nature of the real-life activity was also conveyed through the system, with many evaluation participants strongly contesting the games they took part in. The score display was completely transparent to the user, with no participant questioning its display or what it represented within the game, instead using it to monitor their progress against their opponent, as intended. We can therefore conclude that the aesthetic goal, competition, highlighted through the MDA framework was achieved (user requirement 3), and that the score display interface was designed well in relation to its integration with the rest of the system (usability requirement 5).

The final issue of note relates to the systems functionality on systems with inadequate system specifications. The requirements specified that the system should be functional on systems of a medium to high end nature with high-end graphics capabilities. A range of computers were tested, with high-end computers coping with the demands placed on them by the system. Computers
with a slightly lower system specification did however start to struggle with rendering the game in real-time. Functionality of the system was removed to test what specific system demands were causing the slowdown, with the problem found to be the number of physical objects represented in the game world. This was evidently leading to very large overheads in the collision detection mechanism and subsequently the visibly slower frame rate of the game. To achieve this requirement in future iterations the game would need to find a more efficient way of calculating the potential collisions, perhaps developing an entirely new function to that offered by the ODE physics engine.

7.1. Future Improvements

In a project of this nature, future improvements will always exist as games can always improve upon the interaction experience. We will therefore look at feasible enhancements to this project that can be considered as logical progressions, rather than identifying every possible element that a game of this type could include.

The most logical improvements would include the implementation of the high priority requirements that were unable to be achieved. These include:

- The implementation of all major and minor rules of snooker, in compliance with the official rules outlined by the WPBSA (29). More specifically, rules not featured in the final system, including recording when the ‘miss’ and ‘free ball’ rule states have been reached. Also improving the functionality of partially implemented rules, such as the option of moving the cue ball within the D after the cue ball has been potted. Currently this only operates without considering the position of the camera, and so the movement can potentially be confusing to users.

- Spin effects. Research pertaining to the related ball physics would need to be carried out, as well as looking into possible implementation mechanisms. How this could integrate into the current physical structure of the system would also need to be considered, with the appropriate interface components being integrated, although those specified in the design section of this project could be explicitly utilised.

- Texture mapping could be utilised to further enhance the visual presentation of the game. Of most importance would be visually representing the table markings such as the baulk lines and spot markers, as the user may need to utilise this information when considering what shot to play. I.e. will a ball re-spot in its usual location or will it be re-spotted to a different position due to another ball occupying this location.

- Improving the ball collision mechanism. ODE provides a solid collision detection mechanism, but does not provide as precise a mechanism as we would prefer, as the game relies very heavily on the accuracy of the collision response calculated. Functionality could be included which utilising the overlap or intersection testing collision mechanisms.

Other natural progressions include the implementation of aesthetic features such as sound effects, and perhaps, looking to more distant versions of the game, the possibility of an AI opponent.
7.2. Concluding Remark

The game produced in this project was generally very successful. Many of the requirements were fulfilled, with user feedback on aspects of the system very positive. A better understanding of the problems faced during the development of an interactive game have also been realised, these ranging from the more evident issues, such as how we can implement the game physics to achieve an accurate level of realism, through to more specific hardware concerns, such ensuring the game can be run by a wide array of computer specifications, the latter proving an issue of difficulty. The game produced may not be near the level of a commercially available alternative, but what it does provide is an interaction experience that users can gain a sense of achievement and pleasure from. This leads to the conclusion that a successful game is not reliant on the number of features it can offer the user, but its success is ultimately down to the standard of functionality exhibited by the key features of the game and the overall interaction experience it provides.
Chapter 8

Bibliography


8.1. References


Appendix A

Evaluation of Existing Snooker Games

A.1. World Snooker Championship 2005 (WSC)

*Developed by Blade Interactive. Published by Sega.*

WSC is, as you would expect from a commercially available game, a very comprehensive game. Every aspect of the game is covered in an extensive level of detail.

The depth of the games options are vast and the gameplay wholly representative of the game of snooker. The game comes with numerous modes of play, including Snooker, Pool, Billiards, as well as some more obscure games such as Bar Billiards and Snooker Extra. Also featured is a career mode, in which you can start at the bottom of the ladder and try to work your way through the rankings, which gives the game more focus than simply playing one-on-one matches. In addition there is a practice mode to allow beginners to learn the basics of the game, providing a good way into the game for new users.

The visuals are generally impressive, lighting on the balls realistically rendered and the player models of the most famous players doing a good job of representing their real-life counterparts. However, compared to other commercial products the player models can look fairly unrealistic with some players barely sharing a passing resemblance to their real life counterpart. Sound effects and music both add to the game experience, without interfering with the focus of the game. Commentary by several well-known commentators such as Dennis Taylor and John Virgo creates a more realistic game atmosphere.

The representation of the real-life game is extremely well developed. The game physics are handled very well, with collisions and ball speed feeling very close to how the balls would react in real life. All the rules from snooker have
been implemented, including the more specific rules such as the ‘miss’ rule, which can occasionally play an important part in a game of snooker.

Interaction is dealt with well, providing the user with numerous camera angles to view the table from many different angles. The control system is cleverly implemented considering the number of features available to the user. This allows spin, cue angle and camera angles to be changed in a relatively simple manner, without the controls feeling like a hindrance to the usability of the game itself.

The game is by no means faultless, letting itself down in a number of key design areas. The first, and possibly worst, aspect is its menu interface. It utilises a system that hinders the users’ visibility of currently available options, presenting the visible options in a very confusing circular form. These options are also represented by unclear picture icons, so the user does not know what the unselected options represent until they are navigated to. To get to some of the options you are forced to trawl through a number of others, possibly not knowing that the option you are looking for is on the current submenu as, as mentioned before, not all the menu items are visible onscreen at once. A visually simpler front-end would eradicate many of these problems.

The second problem is the difficulty of the game. Snooker is not the easiest game to represent in terms of its difficulty; you either select the correct angle for the pot and it goes in or it does not. This can make some shots that are very difficult in real life unbelievably easy in the game. Similarly the AI for real life players, for example Ronnie O’Sullivan (World Number 1 in real life), have been programmed to pot almost every shot. This damages the realistic feel of the game, with near miraculous shots being successfully played by your opponent, leaving the user knowing that they also must try to successfully pot every shot, which is not in parallel to how the real game is played. This is an area that is extremely difficult to perfect and no snooker game to date has achieved realistically.

The only other minor problems are with less obvious features. Although the visuals are very good, the table surface itself is visually very basic, the lines showing the baulk end of the table looking slightly out of place with the rest of the visuals. The scoring system sometimes calculates incorrectly in certain situations and the game commentary sometimes seems to relate to a completely different game, although as mentioned before the commentary is a good addition that could be refined in future releases.

Overall the game is probably the best representation of snooker available at the present time. It is a shame that a few needless flaws have found their way into the final product.

Advantages

• Game Visuals. Balls are beautifully rendered and the way they move very realistic. Player models are reasonable, but when compared to other commercial products look fairly basic and blocky.
• Game Physics. Balls react exactly as you would expect in the real game, with collisions, spin and friction all implemented impeccably.
• Sound. The music, effects and the commentary are good features that all add to the gameplay.
• Rule Implementation. Every rule in snooker is implemented in the game, including the full implementation of the miss rule.
• Large array of game modes and features have been implemented. Can even create your own player in career mode.

Disadvantages

• Poor pre-game menus. Available options aren’t clear, which makes navigation extremely difficult. These could benefit from being much more simplified.
• Game difficulty does not realistically relate to the real-life game. Some shots that are virtually unplayable in real-life can be played with relative ease in the game.
• Commentary can be inaccurate, relating to events that have not taken place.
• Some rare erroneous score calculations can occur.

A.2.FooBilliard
- Developed by Florian Berger.

FooBilliard is a well presented representation of the game of pool, clearly implementing the key features successfully.

It has no menu system and no explanation of the controls, which would be useful for using the more advanced functions such as putting spin on the ball which are implemented but have to be found through trial and error of keyboard entry.

Ball physics are realistic, collisions and spin both acting as expected as does the speed of how the balls roll around on the table. With pool being a simpler
game to implement in terms of its rules, it copes with basic constraints such as the cue ball being potted and fouls.

The visuals are impressive for a freeware development, utilising texture mapping for most of the models. It uses a visual power bar for the force with which the cue ball is hit, which fits into the system well without interfering with the rest of the game. The ball visuals could be better, as there is no clear attempt at realistic lighting such as specular highlights, but instead a simple static texture applied to each ball.

Controls are simple, but unclear with no help system implemented to tell you what the main controls are. The camera control is also simple, but what has been implemented has been done well, although different viewpoints would perhaps provide a better means of interaction with the game. One problem with the camera is that there are no boundaries when zooming, allowing the user to actually see through the ball when zooming very close.

These are very minor issues, and the gameplay itself is very good. All of the features key to the game are fulfilled correctly and competently, and is therefore a very enjoyable game to play. All that is clearly lacking is any kind of help system for controls, which could prevent users from utilising all of the games features.

Advantages

- Ball physics implemented well, with collisions, spin and friction all realistically represented.
- Graphics look realistic, with the exception of the lighting on balls.
- Simple but enjoyable, showing that if the features chosen to be implemented are indeed implemented well, a lack of other features does not detract too much from the game experience.
- Power ball simple but effective way of representing how much force will be applied to the cue ball.
- Both mouse and keyboard controls available.

Disadvantages

- Controls, although simple are inaccessible, with no help screen available.
- Balls look too static. Perhaps adding more lighting detail to the balls would solve this.
- Lack of any features except the main mode of play.
- Keyboard controls are very inaccurate. Rotating around the table using the keyboard arrows for example is not handled well, as this moves the camera a large distance each press.
Fig A.2. Screenshot of FooBilliard 3.0
Appendix B

Stakeholder Analysis Questionnaires

Referenced here are the cumulated results from the questionnaires carried out as part of the Stakeholder Requirements Analysis. Twelve stakeholders took part in this analysis and the results that follow detail their combined responses, averaging fields where numeric input was requested. The conclusions drawn from these questionnaires are detailed in chapter 3.3.
Stakeholder Analysis Questionnaire

Question 1 – Have you heard of or played the game of snooker/pool/billiards? Please highlight your answer in bold. (If your response is no, please ignore the rest of the questionnaire).

Yes 12 responses
No 0 responses

Question 2 – How familiar would you consider yourself to be with the rules of snooker? Please highlight your answer in bold.

Very Familiar 5
Quite Familiar 4
Unfamiliar 2
Very Unfamiliar 1

Question 3 – Below are some features that could be present in a game representing a game such as snooker. Please indicate on a scale of 1-5 (5 being most important) how important you consider each feature to be:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Responses</th>
<th>Av</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Quality Graphics</td>
<td>1,3,1,4,4,4,1,2,5,1,4,3</td>
<td>2.75</td>
</tr>
<tr>
<td>Ball Collisions</td>
<td>5,4,5,5,5,5,5,3,4,3,4</td>
<td>4.4</td>
</tr>
<tr>
<td>Realistic ball speed (ball movement)</td>
<td>3,5,4,5,5,4,4,5,4,5</td>
<td>4.4</td>
</tr>
<tr>
<td>Correct implementation of rules</td>
<td>5,5,4,5,5,5,5,4,5,5</td>
<td>4.8</td>
</tr>
<tr>
<td>Scoring System</td>
<td>4,4,5,5,5,4,4,3,4,5,5</td>
<td>4.4</td>
</tr>
<tr>
<td>Power Indicator</td>
<td>4,4,5,3,5,4,4,5,3,5,4,2</td>
<td>4.0</td>
</tr>
<tr>
<td>Spin Effects</td>
<td>4,3,4,3,5,3,3,5,2,3,3</td>
<td>3.6</td>
</tr>
<tr>
<td>Camera Views</td>
<td>2,3,4,4,4,3,4,3,5,4,4,4</td>
<td>3.7</td>
</tr>
<tr>
<td>Easy to use control system</td>
<td>4,5,5,5,5,5,5,5,5,5</td>
<td>4.9</td>
</tr>
<tr>
<td>Clear menu systems</td>
<td>2,4,3,4,4,2,3,4,4,4,3,3</td>
<td>3.3</td>
</tr>
<tr>
<td>Sound Effects</td>
<td>1,2,1,1,3,3,1,2,3,2,2,3</td>
<td>2.0</td>
</tr>
<tr>
<td>AI</td>
<td>1,4,3,4,5,4,2,5,4,3,3,3</td>
<td>3.4</td>
</tr>
<tr>
<td>Various modes of play</td>
<td>1,3,3,2,4,2,2,3,4,3,4,4</td>
<td>3.4</td>
</tr>
<tr>
<td>Number of features</td>
<td>4,3,3,2,3,2,3,3,5,2,4,3</td>
<td>3.0</td>
</tr>
<tr>
<td>Help System</td>
<td>2,2,3,3,3,2,4,5,4,3,5,5</td>
<td>3.4</td>
</tr>
</tbody>
</table>

Question 4 – Are there any features not mentioned in the above list that you would like to see featured in a game of this nature? If so, please indicate the importance of the feature (1-5).

Adjusting angle of the cue 3
(Further analysis showed this was also desired by other stakeholders)

Interactive Snooker Tutorial (basic rules, tactics) 4
Appendix C

The Rules of Snooker

These rules are taken from the official website of the World Professional Billiards and Snooker Association WPBSA (29). The rules specified here relate directly to the rules required to play the game, and rules relating to size and position of equipment should be considered in the design process as to how they can be achieved successfully. Note that not all the rules of snooker are written here, as some relate to specific aspects of the game that need not be considered for this project.

C.1. The Standard Table

Dimensions
1. The playing area within the cushion faces shall measure 11 ft 8½in x 5ft 10in (3569mm x 1778mm).

Pocket Openings
2. There shall be pockets at the corners (two at the Spot end known as the top pockets and two at the Baulk end known as the bottom pockets) and one each at the middle of the longer sides (known as the centre pockets).

Baulk-line and Baulk
3. A straight line drawn 29in (737mm) from the face of the bottom cushion and parallel to it is called the Baulk-line, and that line and the intervening space is termed the Baulk.

The "D"
4. The "D" is a semi-circle described in Baulk with its centre at the middle of the Baulk-line and with a radius of 11½in (292mm).

Spots
5. Four spots are marked on the centre longitudinal line of the table:
i. the Spot (known as the Black Spot), 12¾in (324mm) from a point perpendicularly below the face of the top cushion.

ii. The Centre Spot (known as the Blue Spot), located midway between the faces of the top and bottom cushions.

iii. The Pyramid Spot (known as the Pink Spot), located midway between the Centre Spot and the face of the top cushion.

iv. The Middle of the Baulk-line (known as the Brown Spot).

Two other spots used are located at the corners of the 'D'. Viewed from the Baulk end, the one on the right is known as the Yellow Spot and the one on the left as the Green Spot.

C.2. Definitions

1. Frame
A frame of snooker comprises the period of play from the first stroke, with all the balls set as described in Section 3 Rule 2, until the frame is completed by:
(a) concession by any player during his turn,
(b) claim by the striker when only the Black remains and there is more than seven points difference between the scores in his favour,
(c) the final pot or foul when only the Black remains

2. Game
A game is an agreed or stipulated number of frames.

3. Match
A match is an agreed or stipulated number of games.

4. Balls
(a) The White ball is the cue-ball.
(b) The 15 Reds and the 6 colours are the object balls.

5. Pot
A pot is when an object ball, after contact with another ball and without any infringement of these Rules, enters a pocket.

6. Break
A break is a number of pots in successive strokes made in any one turn by a player during a frame.

7. In-hand
(a) The cue-ball is in-hand
   i. before the start of each frame,
   ii. when it has entered a pocket, or
   iii. when it has been forced off the table.
(b) It remains in-hand until
   i. it is played fairly from in-hand, or
   ii. a foul is committed whilst the ball is on the table
8. **Ball in Play**
   (a) The cue-ball is in play when it is not in-hand.
   (b) Object balls are in play from the start of the frame until pocketed or forced off the table.
   (c) Colours become in play again when re-spotted.

9. **Foul**
   A foul is any infringement of these Rules.

10. **Snookered**
    The cue-ball is said to be snookered when a direct stroke in a straight line to every ball on is wholly or partially obstructed by a ball or balls not on. If one or more balls on can be struck at both extreme edges free of obstruction by any ball not on, the cue-ball is not snookered.
    (a) If in-hand, the cue-ball is snookered if it is obstructed as described above from all possible positions on or within the lines of the "D".
    (b) If the cue-ball is so obstructed from hitting a ball on by more than one ball not on
        i. the ball nearest to the cue-ball is considered to be the effective snookering ball, and
        ii. should more than one obstructing ball be equidistant from the cue-ball, all such balls will be considered to be effective snookering balls.
    (c) When a red is the ball on, if the cue-ball is obstructed from hitting different reds by different balls not on, there is no effective snookering ball.
    (d) The striker is said to be snookered when the cue-ball is snookered as above
    (e) The cue-ball cannot be snookered by a cushion. If the curved face of a cushion obstructs the cue-ball and is closer to the cue-ball than any obstructing ball not on, the cue-ball is not snookered.

11. **Spot Occupied**
    A spot is said to be occupied if a ball cannot be placed on it without that ball touching another ball.

12. **Miss**
    A miss is when the cue-ball fails to first contact a ball on and the referee considers that the striker has not made a good enough attempt to hit a ball on.

**C.3. The Game**

1. **Description**
   Snooker may be played by two or more players, either independently or as sides. The game can be summarised as follows:
   (a) Each player uses the same white cue-ball and there are twenty-one object balls - fifteen Reds each valued 1, and six colours: Yellow valued 2, Green 3, Brown 4, Blue 5, Pink 6 and Black 7.
   (b) Scoring strokes in a player’s turn are made by potting Reds and colours alternately until all the Reds are off the table and then the colours in the ascending order of their value.
(c) Points awarded for scoring strokes are added to the score of the striker.
(d) Penalty points from fouls are added to the opponent’s score.
(e) The winner of a frame is the player or side
   i. making the highest score,
   ii. to whom the frame is conceded, or
   iii. to whom it is awarded under Section 3 Rule 13(c).
(f) The winner of a game is the player or side winning most, or the required number of frames
(g) The winner of a match is the player or side winning most games.

2. Position of Balls
   (a) At the start of each frame the cue-ball is in-hand and the object balls are positioned on the table as follows:
      i. the Reds in the form of a tightly-packed equilateral triangle, with the Red at the apex standing on the centre line of the table, above the Pyramid Spot such that it will be as close to the Pink as possible without touching it, and the base of the triangle nearest to, and parallel with, the top cushion.
      ii. Yellow on the right-hand corner of the "D"
      iii. Green on the left-hand corner of the "D"
      iv. Brown on the Middle of the Baulk-line,
      v. Blue on the Centre Spot,
      vi. Pink on the Pyramid Spot, and

![Fig 1. Layout of the Object Balls in Snooker](image)

3. Mode of Play
   The players shall determine the order of play by lot or in any mutually agreed manner.
   (a) The order of play thus determined must remain unaltered throughout the frame, except a player may be asked by the next player to play again after any foul.
   (b) The player or side to strike first must alternate for each frame during a game.
   (c) The first player plays from in-hand, the frame commencing when the cue-ball has been placed on the table and contacted by the tip of the cue, either
      i. as a stroke is made, or
ii. while addressing the cue-ball.
(d) For a stroke to be fair, none of the infringements described below in Rule 10, Penalties, must occur.
(e) For the first stroke of each turn, until all Reds are off the table, Red or a free ball nominated as a Red is the ball on, and the value or each Red and any free ball nominated as a Red, potted in the same stroke, is scored.
(f)  
ii. If a Red, or a free ball nominated as a Red, is potted, the same player plays the next stroke and the next ball on is a colour of the striker’s choice which, if potted, is scored and the colour is then spotted.
ii. The break is continued by potting Reds and colours alternately until all the Reds are off the table and, where applicable, a colour has been played at following the potting of the last Red.
iii. The colours then become on in the ascending order of their value as per Section 3 Rule 1(a) and when next potted remain off the table, except as provided for in Rule 4 below, and the striker plays the next stroke at the next colour on.
(g) If the striker fails to score or commits a foul, his turn ends and the next player plays from where the cue-ball comes to rest, or from in-hand if the cue-ball is off the table.

4. End of Frame, Game or Match
(a) When only the Black is left, the first score or foul ends the frame excepting only if the scores are then equal.
(b) When the conditions in (a) above apply
i. the Black is spotted.
ii. the players draw lots for choice of playing
iii. the next player plays from in-hand, and
iv. the next score or foul ends the frame.

5. Playing from In-hand
To play from in-hand, the cue-ball must be struck from a position on or within the lines of the "D", but it may be played in any direction.

6. Hitting Two Balls Simultaneously
Two balls, other than two Reds or a free ball and a ball on, must not be struck simultaneously by the first impact of the cue-ball.

7. Spotting Colours
Any colour pocketed or forced off the table shall be spotted before the next stroke is made, until finally potted under Section 3 Rule 3(f).
(a) If a colour has to be spotted and its own spot is occupied, it shall be placed on the highest value spot available.
(b) If there is more than one colour to be spotted and their own spots are occupied, the highest value ball shall take precedence in order of spotting.
(c) If all spots are occupied, the colour shall be placed as near its own spot as possible, between that spot and the nearest part of the top cushion.
(d) In the case of Pink and Black, if all spots are occupied and there is no available space between the relevant spot and the nearest part of the top cushion, the colour shall be placed as near to its own spot as possible on the centre line of the table below the spot.
(e) In all cases, the colour when spotted must not be touching another ball.
8. Snookered After a Foul
After a foul, if the cue-ball is snookered, the referee shall state FREE BALL (see Section 2, Rule 16).

(a) If the player next in turn elects to play the next stroke,
   i. he may nominate any ball as the ball on, and
   ii. any nominated ball shall be regarded as, and acquire the value of, the ball on except that, if potted, is shall then be spotted.

(b) It is a foul if the cue-ball should
   i. fail to hit the nominated ball first, or first simultaneously with the ball on, or
   ii. be snookered on all Reds, or the ball on, by the free ball thus nominated, except when the Pink and Black are the only object balls remaining on the table.

(c) If the free ball is potted, it is spotted and the value of the ball on is scored.

(d) If a ball on is potted, after the cue-ball struck the nominated ball first, or simultaneously with a ball on, the ball on is scored and remains off the table.

(e) If both the nominated ball and a ball on are potted, only the ball on is scored unless it was a Red, when each ball potted is scored. The free ball is then spotted and the ball on remains off the table.

(f) If the offender is asked to play again, the free ball call becomes void.

9. Fouls
If a foul is committed, the referee shall immediately state FOUL.

(a) If the striker has not made a stroke, his turn ends immediately and the referee shall announce the penalty.

(b) If a stroke has been made, the referee will wait until completion of the stroke before announcing the penalty.

(c) If a foul is neither awarded by the referee, nor successfully claimed by the non-striker before the next stroke is made, it is condoned.

(d) Any colour not correctly spotted shall remain where positioned except that if off the table it shall be correctly spotted.

(e) All points scored in a break before a foul is awarded are allowed but the striker shall not score any points for any ball pocketed in a stroke called foul.

(f) The next stroke is played from where the cue-ball comes to rest or, if the cue-ball is off the table, from in-hand.

(g) If more than one foul is committed in the same stroke, the highest value penalty shall be incurred.

(h) The player who committed the foul
   i. incurs the penalty prescribed in Rule 10 below, and
   ii. has to play the next stroke if requested by the next player.

10. Penalties
All fouls will incur a penalty of four points unless a higher one is indicated in paragraph (a) to (d) below. Penalties are:

(a) value of the ball on by
   i. striking the cue-ball more than once,
   ii. striking when both feet are off the floor,
   iii. playing out of turn,
   iv. playing improperly from in-hand, including at the opening stroke,
v. causing the cue-ball to miss all object balls,
vi. causing the cue-ball to enter a pocket,

(b) value of the ball on or ball concerned, whichever is higher, by
i. striking when any ball is not at rest,
ii. striking before the referee has completed the spotting of a colour,
iii. causing a ball not on to enter a pocket,
iv. causing the cue-ball to first hit a ball not on,
v. making a push stroke
vi. touching a ball in play, other than the cue-ball with the tip of the cue
as a stroke is made, or
vii. causing a ball to be forced off the table.

(c) value of the ball on or higher value of the two balls concerned by causing
the cue-ball to first hit simultaneously two balls, other than two Reds or a
free ball and a ball on.

(d) A penalty of seven points is incurred if the striker
i. uses a ball off the table for any purpose,
ii. uses any object to measure gaps or distance,
iii. plays at Reds, or a free ball followed by a Red, in successive strokes,
iv. uses any ball other than White as the cue-ball for any stroke once the
frame has started,
v. fails to declare which ball he is on when requested to do so by the
referee, or
vi. after potting a Red or free ball nominated as a Red, commits a foul
before nominating a colour.

11. Play Again
Once a player has requested an opponent to play again after a foul, such
request cannot be withdrawn. The offended, having been asked to play again,
is entitled to
(a) change his mind as to
   i. which stroke he will play, and
   ii. which ball on he will attempt to hit.
(b) score points for any ball or balls he may pot.

12. Stalemate
If the referee thinks a position of stalemate exists, or is being approached, he
shall offer the players the immediate option of re-starting the frame. If any
player objects, the referee shall allow play to continue with the proviso that the
situation must change within a stated period, usually after three more strokes
to each side but at the referee’s discretion. If the situation remains basically
unchanged after the stated period has expired, the referee shall nullify all
scores and re-set all balls as for the start of a frame and
(a) the same player shall again make the opening stroke,
(b) the same order of play shall be maintained.

13. Foul and a Miss
The striker shall, to the best of his ability, endeavour to hit the ball on. If the
referee considers the Rule infringed, he shall call FOUL AND A MISS unless
only the Black remains on the table, or a situation exists where it is impossible
to hit the ball on.
(a) After a foul and a miss has been called, the next player may request the
    offender to play again from the position left or, at his discretion, from the
original position, in which latter case the ball on shall be the same as it was prior to the last stroke made, namely:
   i. any Red, where Red was the ball on,
   ii. the colour on, where all Reds were off the table, or
   iii. a colour of the striker’s choice, where the ball on was a colour after a Red had been potted.

(b) If the striker, in making a stroke, fails to first hit a ball on when there is a clear path in a straight line from the cue-ball to any part of any ball that is or could be on, the referee shall call FOUL AND A MISS unless either player needed snookers before, or as a result of, the stroke played and the referee is satisfied that the miss was not intentional.

(c) After a miss has been called under paragraph (b) above when there was a clear path in a straight line from the cue-ball to a ball that was on or that could have been on, such that central, full-ball, contact was available (in the case of Reds, this to be taken as a full diameter of any Red that is not obstructed by a colour), then:
   i. a further failure to first hit a ball on in making a stroke from the same position shall be called as a FOUL AND A MISS regardless of the difference in scores, and
   ii. if asked to play again from the original position, the offender shall be warned by the referee that a third failure will result in the frame being awarded to his opponent.

(d) After the cue-ball has been replaced under this Rule, when there is a clear path in a straight line from the cue-ball to any part of any ball that is or could be on, and the striker fouls any ball, including the cue-ball while preparing to play a stroke, a miss will not be called if a stroke has not been played. In this case the appropriate penalty will be imposed and
   i. the next player may elect to play the stroke himself or ask the offender to play again from the position left, or
   ii. the next player may ask the referee to replace all balls moved to their original position and have the offender play again from there, and
   iii. if the above situation arises during a sequence of miss calls, any warning concerning the possible awarding of the frame to his opponent shall remain in effect.

(e) All other misses will be called at the discretion of the referee.

(f) After a miss and a request by the next player to replace the cue-ball, any object balls disturbed will remain where they are unless the referee considers the offending player would or could gain an advantage. In the latter case, any or all disturbed balls may be replaced to the referee’s satisfaction and in either case, colours incorrectly off the table will be spotted or replaced as appropriate.
Appendix D

Usability Principles and Guidelines

D.1. Design Principles

Dix et Al (31) specified three categories of design principles; Learnability, Flexibility and Robustness. The design principles of most relevance to this project are described below:

Learnability
• Predictability – The user should be able to predict the effect of future interactions with the system by their interaction experiences.
• Familiarity – For novice users, this measures the correlation between the users existing knowledge and the knowledge required for effective interaction.
• Consistency – The input expressions or output responses should be the same throughout the system, to avoid the user feeling surprised or frustrated as a result of erroneous results.

Flexibility
• Multi-Threading – Ability of the system to support user interaction pertaining to more than one task at a time. I.e. camera controls should be able to be used in combination with one another.

Robustness
• Observability – Ability of the user to evaluate the internal state of the system from its perceivable representation. I.e. Scores and values of shot indicators should be displayed at the interface.
• Responsiveness – How the user perceives the rate of communication with the system.
• Task conformance – The degree to which the system services support all of the tasks the user wishes to perform and in the way that the user understands them.

D.2. Graphic Design Principles

Rabin (2) describes five graphics design principles that must be considered throughout the visual design process.

**Simplicity** – The simplest solutions are generally the easiest to use and most effective. Sometimes, more information and greater impact can be gained from a minimalist approach.

**Consistency** – Users learn through repetitive actions and are more efficient if system behaviour can be predicted. Graphics should be placed consistently to ensure the user is not frustrated when looking for specific system components.

**Know the Target User** – Understanding and predicting how the system will be used by the target audience. This includes use of colour in international projects.

**Colour Usage** – A System should not rely on one colour alone to convey critical information. Colour blindness should also be considered for important visual features. Large amounts of light test on dark background should also be avoided, due to its difficulty to read.

**Feedback Mechanisms** – Visual and auditory elements to help the user understand interaction with the system. This can include sound, rollover buttons, visual prompts and visual highlight changes.
Appendix E

System Screenshots of Developmental Stages

Here we have included stages in the graphical development lifecycle, to show how the system visually evolved through the development process.

*Figure E.1. Initial Table Model with basic material properties and lighting*
Figure E.2. Ball rendering. Shows how we first implemented the representation of one ball, applying lighting to this object, before duplicating this to produce many balls.

Figure E.3. The visual representation and location of balls is completed.
Figure E.4. The final visual representation of the system, with the second score display and power indicator frame present.
Appendix F

Testing and Evaluation

F.1. Defect Testing

F.1.1. Black Box Testing

A comprehensive list of the tests carried out on the system can be found in the tables below. The expected output and actual resultant output were recorded for each test case.

<table>
<thead>
<tr>
<th>No.</th>
<th>Test</th>
<th>Expected output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Press Left arrow key</td>
<td>Moves the aim of the camera left (rotation to the right)</td>
<td>Output as expected</td>
</tr>
<tr>
<td>2.</td>
<td>Press Right arrow key</td>
<td>Moves the aim of the camera right (rotation to the left)</td>
<td>Output as expected</td>
</tr>
<tr>
<td>3.</td>
<td>Hold Shift and press Left arrow key</td>
<td>Moves the aim of the camera left at a slower pace (rotation to the right)</td>
<td>Output as expected</td>
</tr>
<tr>
<td>4.</td>
<td>Hold Shift and press Right arrow key</td>
<td>Moves the aim of the camera left at a slower pace (rotation to the right)</td>
<td>Output as expected</td>
</tr>
<tr>
<td>5.</td>
<td>Press Up arrow key</td>
<td>Zooms the camera view towards the cue ball, until it stops due to boundary restrictions placed on the control</td>
<td>Output as expected</td>
</tr>
<tr>
<td>6.</td>
<td>Press Down arrow key</td>
<td>Zooms the camera view away from the cue ball, until it stops due to</td>
<td>Output as expected</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>Hold A and press Up arrow key</td>
<td>Moves the vertical location of the camera up, thus looking at the cue ball from a greater height, until it stops due to boundary restrictions placed on the control</td>
<td>Output as expected</td>
</tr>
<tr>
<td>8.</td>
<td>Hold A and press Down arrow key</td>
<td>Moves the vertical location of the camera down, thus looking at the cue ball from a lower height until it stops due to boundary restrictions placed on the control</td>
<td>Output as expected</td>
</tr>
<tr>
<td>9.</td>
<td>Hold S and press Left arrow key</td>
<td>Decreases power percentage on the power indicator</td>
<td>Output as expected</td>
</tr>
<tr>
<td>10.</td>
<td>Hold S and press Right arrow key</td>
<td>Increases power percentage on the power indicator</td>
<td>Output as expected</td>
</tr>
<tr>
<td>11.</td>
<td>Press spacebar</td>
<td>Plays a shot</td>
<td>Output as expected</td>
</tr>
<tr>
<td>12.</td>
<td>Press spacebar once to play the shot, then press space an additional number of times</td>
<td>Plays the initial shot upon first press. All further presses have no effect</td>
<td>Output as expected</td>
</tr>
<tr>
<td>13.</td>
<td>Press Tab. Then Press Tab again.</td>
<td>Switch to overhead view. Upon second press switches back to normal view.</td>
<td>Output as expected</td>
</tr>
<tr>
<td>14.</td>
<td>At the start of a frame, or directly after the cue ball has been potted, hold M and press one of the arrow keys</td>
<td>Movement of the cue ball around the D should be controllable</td>
<td>Output as expected. Movement is not relative to camera angle and restricted to a rectangular area within the D as implemented.</td>
</tr>
<tr>
<td>15.</td>
<td>Use a combination of the camera controls to test the boundaries for deadlock</td>
<td>Camera should always be controllable, with no deadlock scenario reached through using a combination of camera movements</td>
<td>Problem found with the combination of vertical movement and zoom. The zoom did not have any height restrictions. This was a problem as vertical movement combined with zoom meant the camera could move back to infinity. Camera also bit jittery when the camera was moved during a shot taking place.</td>
</tr>
</tbody>
</table>
### Table F.2. Physics System Tests

<table>
<thead>
<tr>
<th>No.</th>
<th>Test</th>
<th>Expected output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Ball hits a cushion</td>
<td>The ball should rebound off the cushion back into the area of play at an angle relative to the angle of collision</td>
<td>As expected the ball rebounded off the cushion at a realistic angle.</td>
</tr>
<tr>
<td>2.</td>
<td>Ball hits another ball</td>
<td>The two balls should collide, with a relative directional force exerted on the ball struck.</td>
<td>When high levels of power are applied to the cue ball the collisions become less accurate. A smaller maximum power was applied to rectify this.</td>
</tr>
<tr>
<td>3.</td>
<td>Ball hits pocket knuckle</td>
<td>Ball should rebound off knuckle at an angle relative to where it hit the cushion.</td>
<td>Output as expected</td>
</tr>
<tr>
<td>4.</td>
<td>Ball enters a pocket</td>
<td>Ball should fall into pocket with respect to gravity</td>
<td>Output as expected</td>
</tr>
<tr>
<td>5.</td>
<td>Cue ball is hit with a relative force using the spacebar</td>
<td>The ball should slow down over time, eventually coming to a standstill.</td>
<td>Output as expected</td>
</tr>
<tr>
<td>6.</td>
<td>Cue ball is hit with a relative force using the spacebar into one or more other balls</td>
<td>The ball should collide with the other balls, exerting a relative force on them. All balls should slow down and eventually come to a standstill.</td>
<td>Output as expected</td>
</tr>
<tr>
<td>7.</td>
<td>Potting a ball with a large amount of power applied</td>
<td>The ball should hit the back of the pocket and fall down into the pocket due to gravity</td>
<td>Output as expected</td>
</tr>
<tr>
<td>8.</td>
<td>When a ball is potted, it is removed from the user’s field of view.</td>
<td>The ball should disappear, as its body has been destroyed</td>
<td>Output as expected</td>
</tr>
</tbody>
</table>

### Table F.3. Rule System Tests

<table>
<thead>
<tr>
<th>No.</th>
<th>Test</th>
<th>Expected output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Hit a red ball on the first shot of a player turn</td>
<td>Should recognise that no pot has occurred, and adjust the interface to show that it is now the opposing players turn</td>
<td>Output as expected</td>
</tr>
<tr>
<td>2.</td>
<td>Pot one or more red balls on the first shot of a players turn</td>
<td>Should increment the relevant players score and break by the relevant amount (1 point per red)</td>
<td>Output as expected</td>
</tr>
</tbody>
</table>
potted), allowing the current player to play another shot

3. Hit a coloured ball directly after a red ball has been potted in the same turn

<table>
<thead>
<tr>
<th>Expected output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Should recognise that a legal shot has been played, but no pot has occurred, adjusting the interface to show that it is now the opposing players turn</td>
<td>Output as expected</td>
</tr>
</tbody>
</table>

4. Pot a coloured ball directly after a red ball has been potted in the same turn.

<table>
<thead>
<tr>
<th>Expected output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Should increment the relevant players score by the relevant amount (yellow=2, green=3, brown=4, blue=5, pink=6, black=7), re-spotting the coloured ball potted accurately and allowing the current player to play another shot</td>
<td>Output as expected</td>
</tr>
</tbody>
</table>

5. Pot one or more red balls directly after a colour ball has been potted in the same turn.

<table>
<thead>
<tr>
<th>Expected output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Should increment the relevant players score by the relevant amount (1 per red potted), allowing the current player to play another shot</td>
<td>Output as expected</td>
</tr>
</tbody>
</table>

6. Hitting or potting a legal coloured ball when all red balls have been potted (with accompanying colours as specified by the rules). Colours must be considered legal in the order yellow, green, brown, blue, pink, black

<table>
<thead>
<tr>
<th>Expected output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hitting or potting a colour in this order is considered a legal shot, incrementing the current players score if a legal pot is achieved</td>
<td>Problem found with yellow ball being re-spotted at this stage of the game</td>
</tr>
</tbody>
</table>

### Illegal Shots

<table>
<thead>
<tr>
<th>No.</th>
<th>Test</th>
<th>Expected output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.</td>
<td>Pot the cue ball directly, without hitting another ball</td>
<td>Should increment the opposing player’s score by 4 points and re-spot the cue ball in the D. Should adjust the interface to show that it is now the opposing players turn</td>
<td>Output as expected</td>
</tr>
</tbody>
</table>

8. Pot the cue ball after hitting another ball on the table

<table>
<thead>
<tr>
<th>Expected output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Should increment the opposing players score by the relevant amount (red, yellow, green, brown = 4, blue=5, pink=6, black=7)</td>
<td>Error found when colour was the legal ball, foul was not correctly scored</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>9.</td>
<td>Hit a coloured ball when a red ball is the legal ball</td>
</tr>
<tr>
<td>10.</td>
<td>Hit a red ball when a coloured ball is the legal ball</td>
</tr>
<tr>
<td>11.</td>
<td>Pot a coloured ball when a red ball is the legal ball.</td>
</tr>
<tr>
<td>12.</td>
<td>Hitting or potting an illegal coloured ball when all red balls have been potted (with accompanying colours as per the rules). Colours must be considered legal in the order yellow, green, brown, blue, pink, black.</td>
</tr>
<tr>
<td>13.</td>
<td>More than one coloured ball is potted in one shot, when an illegal ball is hit first</td>
</tr>
<tr>
<td>14.</td>
<td>Potting a red and a</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output as expected
coloured ball within the same shot opposing players score by the relevant amount, re-spotting the potted colours accurately and adjusting the interface to show that it is now the opposing players turn

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15.</td>
<td>Potting the cue ball and a coloured ball within the same shot.</td>
<td>Should increment the opposing players score by the relevant amount, re-spotting both balls accurately and adjusting the interface to show that it is now the opposing players turn</td>
</tr>
</tbody>
</table>

The scoring system was subject to further defect testing, to ensure the correct calculation of player scores during a game. The first part of this defect testing is shown in the table below. Only successful pots or fouls are detailed, rather than the score after every turn. The current break output is also detailed by B=x.

Table F.4. Score Tests

<table>
<thead>
<tr>
<th>Action</th>
<th>Expected score</th>
<th>Actual score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player 2 pots red</td>
<td>P1=0, P2=1 (B=1)</td>
<td>P1=0, P2=1 (B=1)</td>
</tr>
<tr>
<td>Player 2 pots red</td>
<td>P1=0, P2=2 (B=1)</td>
<td>P1=0, P2=2 (B=1)</td>
</tr>
<tr>
<td>Player 2 pots red</td>
<td>P1=0, P2=3 (B=1)</td>
<td>P1=0, P2=3 (B=1)</td>
</tr>
<tr>
<td>Player 2 pots pink (same break)</td>
<td>P1=0, P2=9 (B=7)</td>
<td>P1=0, P2=9 (B=7)</td>
</tr>
<tr>
<td>Player 1 pots red</td>
<td>P1=1 (B=1), P2=9</td>
<td>P1=1 (B=1), P2=9</td>
</tr>
<tr>
<td>Player 1 pots blue (same break)</td>
<td>P1=6 (B=6), P2=9</td>
<td>P1=6 (B=6), P2=9</td>
</tr>
<tr>
<td>Player 2 pots red</td>
<td>P1=6, P2=10 (B=1)</td>
<td>P1=6, P2=10 (B=1)</td>
</tr>
<tr>
<td>Player 2 pots blue (same break)</td>
<td>P1=6, P2=15 (B=6)</td>
<td>P1=6, P2=15 (B=6)</td>
</tr>
<tr>
<td>Player 2 pots red (same break)</td>
<td>P1=6, P2=16 (B=7)</td>
<td>P1=6, P2=16 (B=7)</td>
</tr>
<tr>
<td>Player 2 pots blue (same break)</td>
<td>P1=6, P2=21 (B=12)</td>
<td>P1=6, P2=21 (B=12)</td>
</tr>
<tr>
<td>Player 2 pots red (same break)</td>
<td>P1=6, P2=22 (B=13)</td>
<td>P1=6, P2=22 (B=13)</td>
</tr>
<tr>
<td>Player 1 pots red</td>
<td>P1=7 (B=1), P2=22</td>
<td>P1=7 (B=1), P2=22</td>
</tr>
<tr>
<td>Player 2 pots red</td>
<td>P1=7, P2=23 (B=1)</td>
<td>P1=7, P2=23 (B=1)</td>
</tr>
<tr>
<td>Player 1 pots red</td>
<td>P1=8 (B=1), P2=23</td>
<td>P1=8 (B=1), P2=23</td>
</tr>
<tr>
<td>Player 1 pots green</td>
<td>P1=11 (B=4), P2=23</td>
<td>P1=11 (B=4), P2=23</td>
</tr>
<tr>
<td>Player 2 pots cue ball</td>
<td>P1=15, P2=23</td>
<td>P1=15, P2=23</td>
</tr>
</tbody>
</table>

No irregular scoring patterns were observed in this testing.

F.1.2. White Box Testing

White box testing was carried out on the most important algorithms in the system. Below we outline the functions tested, what was tested, the expected output, and the actual output.
F.1.2.1. Function: getBallState

Test: The cue ball was hit into a ball(s). The function was followed to ensure the correct ball was identified in the function, as well as identifying when all the balls on the table were stationary.

Must be tested with:
1. Red balls
2. Colour balls.

Expected: The cue ball should be recognised as moving and number of stationary balls should therefore be one less than the number in play, thus returning 1. The location of the cue ball should be updated into the variable ballPos[][].

Upon hitting a ball(s), the number of stationary balls should decrease by the number moving onscreen, and locations of moving balls should be updated. Then as they come to a standstill, the number of stationary balls should become equal to the number of balls in play, and the function should return 0.

Actual: As expected, ballPos was updated when a ball was discovered to be moving, and the function returned 1. When all balls stopped the function returned 0.

F.1.2.2. Function: calculateScores

Test: Should test score scenarios for:
1. No ball hit
2. Cue ball potted without hitting another ball
3. Red ball is legal ball and colour ball potted
4. Red(s) potted legally
5. Colour ball is legal ball and red ball potted
6. Two colours potted
7. Legal ball potted when no red left in play
8. Illegal colour ball potted when no red balls left in play
9. Colour ball potted legally
10. Colour ball not potted after a red has been in same turn
11. Illegal ball hit first
12. Illegal ball hit first and cue ball potted
13. Two balls potted after illegal ball being hit

Expected: We will list expected variable outputs for testing. The key variable at each stage are score, foul and foulState although are specified where required by the scenario.

1. ballHit = 0, score = 0, foul = 4, foulState = 1
2. ballHit = 0, score = 0, foul = 4, foulState = 1, cueBallPotted = 1
3. score = 0, foul = relative to colour potted, foulState = 1
4. ballHit = 1, score = relative to number of reds potted, foul = 0, foulState = 0, ballToPot = 8
5. score = 0, foul = foul relative to colour hit, foulState = 1
6. score = 0, foul = foul relative highest value colour potted, foulState = 1
7. score = score relative to colour potted, foul = 0, foulState = 0
8. score = 0, foul = foul relative to colour potted, foulState = 1
9. score = score relative to colour potted, foul = 0, foulState = 0
10. should change ballToPot back to 1 if reds left, else ballToPot = 2
11. score = 0, foul = foul relative to ball hit, foulState = 1
12. score = 0, foul = foul relative to ball hit, foulState = 1
13. score = 0, foul = highest value foul value when the pot foul and the ball
hit foul are compared, foulState = 1

Actual: The scenarios outputs were:
1. to 6. as expected
7. not as expected – counted as foul shot
8. to 13. as expected

F.1.2.3. Function: checkPotted

Test: Check the output when:
1. One red ball is potted
2. Two or more red balls are potted
3. A colour ball is potted
4. Two colours are potted
5. The cue ball is potted

Expected:
1. When a red ball is potted should return colourPotted = 1, update the
variable offTable for that red ball and destroy the body associated to the
appropriate red ball. ballsInPlay should be decremented by 1.
2. When a red ball is potted should return colourPotted = Number of reds
potted, update the variable offTable for these red balls and destroy the
bodies associated to them. ballsInPlay should be decremented by
number of red balls potted.
3. When a colour ball is potted, colourPotted should reference the relevant
colour (2-7 for yellow-black).
4. When a colour ball is potted, colourPotted should reference the relevant
foul index (8-11) dependent on the combination of balls potted.
5. colourPotted = 1 to show cue ball has been potted

Actual:
1. colourPotted = 1. Body of red ball destroyed. ballsInPlay was
decremented and offTable updated appropriately.
2. colourPotted = number of reds potted. Body of red balls destroyed.
ballsInPlay was decremented and offTable updated appropriately.
3. colourPotted = relevant colour. Was tested with all colours and outputs
were all as expected
4. colourPotted = relevant foul value. Was tested with different
combinations and outputs were all as expected
5. colourPotted returned 1

F.1.2.4. Function: spotClear

Test: Test each ball that can be re-spotted. This includes all colours and the cue
ball.

Expected: Function should test every ball on table against the re-spot location, and
return if clear or not. If clear then re-spot the ball. If not should move
location towards the top cushion until a clear location is found. Upon finding a location, the ball should be re-spotted, and ballPos array updated with these location details.

Actual: Function tests locations for colours and cue ball correctly. If location is clear it re-spots the ball, if not moves the location towards the baulk cushion which is incorrect. ballPosition was then updated when a clear location is found, and the ball is re-spotted.

This was corrected by specifying correct movement of the new spot location to ensure it moved towards the top cushion.

F.2. Integration Testing

F.2.1. Friction Mechanism

Test: Hit any number of balls on the table, to test the friction mechanism

Expected: When the applyFriction Function is called, check that for each ball damping is applied (applyDamping function) and then from this that the velocityScale function is called and returns the relative scale of damping for that ball.

Actual: Occasionally finding that balls were still moving by a very small degree, even though a velocity of 0 was being applied to the ball, which in rare situations meant the system was unable to recognise the end of a players turn. This was accounted as an error in the ODE functionality and code to allow for the end of a turn being recognised in these circumstances was implemented.

F.2.2. Camera functionality

Test: Move the camera by using the rotation, zooming and vertical movement controls

Expected: The movement carried out should be updated in the object variables (xPos, yPos, zPos, xView, yView, zView and camR) and then called back when setCamera is called.

Actual: All camera variables are updated through the specific camera functions. When set camera is called, these updated variables are referenced and used to set the viewport with the gluLookAt function.

F.2.3. Render Function

Test: Play a shot. Test that the Render function checks for ball states and balls potted calls these functions correctly and receives the relevant values back.
Expected: Should call the functions that setup the OpenGL graphics (lights, table and balls). Should then check the ball state, calling the functions that apply friction to the ball when the balls are found to be moving. When the balls have stopped (ballState = 0), the checkPotted function is called to set variables relating to whether (1) the cue ball has been potted, (2) any red balls potted, and (3) any coloured balls potted. The calculateScore function is passed these three variables and should return a value stating if the player made a legal pot (1=legal pot, 0 otherwise). If no pot is made then the variable 'player' is changed to the opponent (player1 = 1, player2 = 2).

Actual: The function operates as expected. When a ball is potted, the relevant variables are updated (cueBallPotted, redPotted, colourPotted). The calculateScore function

F.3. Evaluations

F.3.1. Cooperative Evaluation

Cooperative evaluation was performed with several test users. These evaluations were conducted with the test user playing against the evaluator. The evaluator ensured that the user was confirming what they expected of the system, as well as detailing what they thought of any important aspects of the game. Any unexpected output was also recorded. An outline of the points raised during one such evaluation is detailed below.

Participant: Nicholas Steel
User Background: User is familiar with snooker and the rules of the game.

- User was asked to use controls to view table and comment on its usability. User liked the targeting system, stating that its use within the system was apparent by the aim line displayed on screen.
- User immediately comments that they like the layout of the system. User also maximises window upon start-up.
- User is asked to play the first shot. Finds controls easy to use, and uncomplicated.
- User interacts with system for several turns. States that a feature to view the table from any perspective would be useful, perhaps using a first person ‘walk-around’ feature in which the user can virtually move around the table.
- When asked about the realism of movement, the user is happy with the game speed implemented, although comments that a spin feature would add to the depth of the game options.
- Notes that power options could utilise shift to specify very precise levels of power.
- Finds that collisions at high speeds when the aim is towards the edge of the target ball can sometimes produce erroneous collisions.
• At one point in the test, the user expects to obtain a ‘free ball’. The system does not provide this rule implementation and so the user is left confused and having to deal with this lack of functionality.
• User expresses opinion that maybe the aim line could be present on overhead view, whilst also commenting that this is objective, as it could make the game easier.

F.3.2. Think-Aloud evaluation Script

As stated in the testing chapter, several users were asked to participate in ‘think-aloud’ evaluations of the system. A transcript of one evaluation is drawn up here, from an evaluation with two test users playing against one another. Their main points are detailed here and reviewed in the testing section.

Key
Participant 1 - Josh Ackland - JA
Participant 2 - Victor Gonzalez-George - VG
Evaluator – Ashley Nolan – AN

Participants both have some knowledge of the game of snooker.

• Users are set up with the system and the control system explained to them.

VG – “I’m not exactly sure how the aiming system works”
JA – “The line through the middle of the cue ball shows where the ball is being aimed”

• VG plays first shot

VG – “The ball movements seem pretty realistic and better when compared to some of the freeware games I’ve played”
JA – “Yes, I like the way the game seems to react quite well in respect to the speed of the balls movement”
VG – “It would be useful to have spin as a shot option though.”
AN – “Would this affect any decision to play the game more or less?”
VG – “No. In a way it does make the game less complicated, but if the feature was available, the variety of shots I could play would be a lot bigger”.

• JA pots first red ball

JA – “I was expecting some sort of prompt to nominate, but I’m going to just aim to pot the blue now”.

• JA misses the blue and hits pink

JA – “I’m a bit confused as I aimed for the blue and hit the pink but no foul was called due to the lack of a nomination feature. Perhaps implementing some sort of nomination feature would stop this kind of unrecognised foul.”
• Several shots are played by each player. Note that they seem to be getting more used to the control

VG – “I think a power bar might be a better interface than the percentage indicator.”
JA – “I like the percentage indicator, as it gives a more precise visual representation of the amount of power you will put on the shot.”
VG – “I do like the use of shift for more specific aiming; the keyboard control is sensibly placed for this feature. The controls for power increase and decrease could be better placed on the up/down keys rather than the left/right keys. I’m getting more used to the aiming system now though, it just a case of getting to grips with it I think.”
AN – “So what do you think about the games graphics?”
JA – “I like them, they are simple, and give the feel of a game of snooker, which is obviously important! If playing a web game of snooker accessibility and ease of use would be most important to me.”
VG – “…and pick up and play.”
AN – “What other features would add depth to the game?”
VG – “Sound effects would add depth to the game and an element of realism. Perhaps a view to walk around the table too.”
AN – “Would you be likely to use a walk around view?”
VG – “Not often, but it could add depth to the game options.”
JA – “AI could be a good feature, but I quite like playing games like this on my own to get high breaks.”

• VG plays a shot and the ball hits the pocket knuckles and rebounds back into play

VG – “The pockets act realistically to balls hitting the knuckles. I think I’d rather have a physical cue displayed behind the ball as well as the aim line to make it clearer.”
JA – “I like the aim line as it is, but maybe on the overhead view, some kind of aim could be displayed.”
AN – “If spin could be implemented as a button for backspin, topspin and stun, would this be a suitable option, or would it seem too compromised?”
VG – “I think it would be better than having no option of spin at all.”
JA – “I quite like the games that use the mouse for power control, moving the mouse forward to take a shot.”

AN – “Just one final question, did you notice at all, a lack of baulk line on the table?”
VG – “No, but that would be a good addition to system.”
JA – “No, but yes it would be good to see in the game.”
Appendix G

Program Listings

Functions

The following functions can be found listed in this section

- setSpotLocations
- SetupProjection
- applyForce
- getBall
- applyFriction
- applyDamping
- velocityScale
- getBallState
- checkPotted
- spotClear
- calculateScore
- foulIndex
- setCamera
- moveCueball
- Render
- RenderScores
- RenderAimLine
/**
 * setSpotLocations: sets the spot locations for respots
 */
void CGfxOpenGL::setSpotLocations()
{
    //sets the locations of the cue ball at the start, and colour spots
    //these are then referenced when respotting balls
    //white ball
    spotPositions[0][0] = -15;
    spotPositions[0][2] = -122;
    //yellow
    spotPositions[1][0] = -30;
    spotPositions[1][2] = -122;
    //green
    spotPositions[2][0] = 30;
    spotPositions[2][2] = -122;
    //brown
    spotPositions[3][0] = 0;
    spotPositions[3][2] = -122;
    //blue
    spotPositions[4][0] = 0;
    spotPositions[4][2] = 0;
    //pink
    spotPositions[5][0] = 101.5;
    spotPositions[5][2] = 101.5;
    //black
    spotPositions[6][0] = 0;
    spotPositions[6][2] = 170;
    //all y-values are the same
    for (int i = 0; i<7; i++)
    {
        spotPositions[i][1] = 3.130599976;
    }
}

/**
 * SetupProjection: sets up the main gameview projection
 */
void CGfxOpenGL::SetupProjection(long width, long height)
{
    if (height == 0) //don't want a divide by zero
    {
        height = 1;
    }
    //MAIN VIEWPORT
    glViewport(0, 0, width, height);  //reset the viewport to new dimensions
    glMatrixMode(GL_PROJECTION);   //set projection matrix
    glLoadIdentity();      //reset projection matrix
    //calculate aspect ratio of window
    gluPerspective(56.0f,(GLfloat)width/(GLfloat)height,0.1f,1000.0f);
    glMatrixMode(GL_MODELVIEW);    //set modelview matrix
    glLoadIdentity();      //reset modelview matrix
    glClear (GL_DEPTH_BUFFER_BIT);   //set depth buffer
    //m_windowWidth = width;
    //m_windowHeight = height;
}

/**
 * applyForce: Receives the percentage power to apply
 * and the camera location to know the direction to apply the force
 */
void CGfxOpenGL::applyForce (float percentageForce, CCamera camera)
{
    const dReal * cueBallReal = dBodyGetPosition(cueBall); //get position of cueball
    //setup variables
    float x = 0.0;
    float z = 0.0;
    float xLength, zLength, xPercentage, zPercentage, total = 0;
    //find percentage of maximum force
    float force = (35000000 / 100) * percentageForce;
    if (camera.xPos - cueBallReal[0] == 0)
        z = force; //if camera lies along x axis
    else if (camera.zPos - cueBallReal[2] == 0)
        x = force; //if camera lies along z axis
    else
    {
        //works out a percentage of the x and z lengths and applies force relatively
        xLength = camera.xPos - camera.xView;
        zLength = camera.zPos - camera.zView;
        //if negative x or z value, change to positive for calculations
        if (xLength < 0) {
            xLength = -xLength;
        }
    }
if (zLength < 0) {
    zLength = -zLength;
}

total = xLength + zLength; //total length

//percentage force in x and z directions
xPercentage = (xLength / total) * 100;
zPercentage = (zLength / total) * 100;

x = (force / 100) * xPercentage;
z = (force / 100) * zPercentage;

if (camera.xPos > cueBallReal[0]) x = -x;
if (camera.zPos > cueBallReal[2]) z = -z;
}

DBodyAddForce(cueBall, x, 0, z); //add force to cueball

/**
getBall: returns value of the ball relative to the body received
**/
int CGfxOpenGL::getBall(dBodyID body)
{
    for (int i = 0; i<15; i++)
        {
            if (body == redBall[i])
                return 1;
        }
    for (int j = 0; j<6; j++)
        {
            if (body == colourBall[j])
                {
                    switch(j)
                    {
                    case 0:
                        return 2;
                    case 1:
                        return 3;
                    case 2:
                        return 4;
                    case 3:
                        return 5;
                    case 4:
                        return 6;
                    case 5:
                        return 7;
                    }
                }
        }
}

/**
applyFriction: Apply friction to every ball on table
**/
void CGfxOpenGL::applyFriction()
{
    applyDamping(cueBall, 0);
    for (int i = 0; i<15; i++)
        {
            if (offTable[i+1] == 1)
                {
                    applyDamping(redBall[i], i+1);
                }
        }
    for (int j = 0; j<6; j++)
        {
            if (offTable[j+16] == 1)
                {
                    applyDamping(colourBall[j], j+16);
                }
        }
}

/**
applyDamping: Called by applyFriction
receives scale of damping from velocityScale
and multiples ball velocity by relative amount
**/
void CGfxOpenGL::applyDamping(dBodyID body, int bodyNumber)
{
    const dReal * ballVelocity = dBodyGetLinearVel(body);
    //Damping applied...
    float vScale = velocityScale(body, bodyNumber);
    if (vScale == 0)
        {vScale = 0.995;}}
vel[1] = ballVelocity[1] * 0.994;
dBodySetLinearVel(body, vel[0], vel[1], vel[2]);
} else if (vScale == 1)
{
    vel[0] = ballVelocity[0] * 0.99;
    dBodySetLinearVel(body, vel[0], vel[1], vel[2]);
} else if (vScale == 2)
{
    vel[0] = ballVelocity[0] * 0.98;
    vel[1] = ballVelocity[1] * 0.98;
    dBodySetLinearVel(body, vel[0], vel[1], vel[2]);
} else if (vScale == 3)
{
    dBodySetLinearVel(body, 0, 0, 0);
}

/**
velocityScale: Called by applyDamping function
returns scale of damping to be applied
**/
float CGfxOpenGL::velocityScale(dBodyID body, int bodyNo)
{
    const dReal * ballPosition = dBodyGetPosition(body);
    float xDiff = ballPosition[0] - ballPos[bodyNo][0];
    float zDiff = ballPosition[2] - ballPos[bodyNo][2];
    if (xDiff < 0)
    {
        xDiff = -xDiff;
    }
    if (zDiff < 0)
    {
        zDiff = -zDiff;
    }
    if ((xDiff < 0.005) && (zDiff < 0.005))
    {
        return 3;
    }
    else if ((xDiff < 0.1) && (zDiff < 0.1))
    {
        return 2;
    }
    else if ((xDiff < 0.15) && (zDiff < 1))
    {
        return 1;
    }
    return 0;
}

/**
getBallState: function checks for the end of a shot
if returns 0, all balls have stopped
if i, some are still moving and damping should be reapplied
**/
int CGfxOpenGL::getBallState()
{
    int ballsStill = 0;
    const dReal * position;
    for (int i = 0; i<22; i++)
    {
        if (offTable[i] != 1)
        {
            if (i == 0) {
                position = dBodyGetPosition(cueBall);
            } else if (i > 15) {
                position = dBodyGetPosition(colourBall[i-16]);
            } else {
                position = dBodyGetPosition(redBall[i-1]);
            }
            if (ballPos[i][0] > position[0] - 0.00005) && (ballPos[i][0] < position[0] + 0.00005) &&
                (ballPos[i][2] == position[2])
            {
                ballsStill++;
            } else {
                ballPos[i][0] = position[0];
                ballPos[i][1] = position[1];
                ballPos[i][2] = position[2];
            }
        }
    }
    return ballsStill;
}
if (ballsStill == ballsInPlay) {
    return 0;
} else {
    return 1;
}

/**
 * checkPotted: check for what balls have been potted by checking y value (ballPos[x][1])
 * the int that is passed to the function represents
 * 1: CuBall
 * 2: Red Balls
 * 3: Colours
 **/ 
int CGfxOpenGL::checkPotted(int ballType) 
{
    int colourPotted = 0;
    switch (ballType) {
        case 1: //cuball checking
            if (ballPos[0][1] < 0) {
                //CuBall has been potted
                //Respot white
                respot(0);
                //Update Score: 4 to opponent if going for red ball
                //    more if going for blue(5), pink(6) or black(7)
                return 1; //signal foul shot
            } else return 0;
            break;
        case 2: //redball checking
            for(int i=0; i<15; i++) {
                if ((ballPos[i+1][1] < 0) && (offTable[i+1] != 1)) {
                    //a red has been potted
                    offTable[i+1] = 1;
                    dGeomDestroy(redBallGeom[i]); //destroy geom
                    colourPotted = colourPotted++; //update colourPotted
                    ballsInPlay = ballsInPlay--; //update ballsInPlay
                }
            }
            return colourPotted;
            break;
        case 3: //loop through each checking for potted colours
            if (offTable[i+16] != 1) {
                //will either return 2-7 if one colour potted
                //if two balls potted will return:
                // 8: 4 point foul
                // 9: 5 point foul
                //10: 6 point foul
                //11: 7 point foul
                //check each of the six colours
                for (int i=0; i<6; i++) {
                    if ((ballPos[i+16][1] < 0) && (offTable[i+16] != 1)) {
                        colourPotted = i+2;
                        if (onColours != 1) //if reds still left, respot
                            respot(i+1);
                    }
                }
                //colour potted
                //just need to check that it is the only colour potted now
                for (int j=i+1; j<6; j++) {
                    if (ballPos[j+16][1] < 0) {
                        if (j==1) || (j==2) {
                            colourPotted = 8;
                        } else {
                            colourPotted = 9;
                        }
                    } else {
                        colourPotted = 10;
                    }
                    if (j == 5) {
                        colourPotted = 11;
                    }
                }
            }
            break;
    }
    return index relating to colours potted
return colourPotted;
}
return 0;
break;
default:
return 0;
break;
}
/**
respot: takes the number of the ball to be respotted and respots it if position is free
**/
void CGfxOpenGL::respot(int ballNumber)
{
dVector3 tempPos;
bool respotted = false; //identifies when the ball has been respotted
tempPos[0] = spotPositions[ballNumber][0];
tempPos[1] = spotPositions[ballNumber][1];
tempPos[2] = spotPositions[ballNumber][2];
if (ballNumber == 0) //cuesball respot
{
while (respotted == false)
{
dBodySetPosition (cueBall,tempPos[0],tempPos[1],tempPos[2]);
ballPos[0][0] = tempPos[0];
bhallPos[0][1] = tempPos[1];
bhallPos[0][2] = tempPos[2];
if (spotClear(ballNumber))
{
respotted = true;
}
else
{
//need allowance for balls to go on other unoccupied spots
tempPos[0] = tempPos[0] - 1;
}
}
else //colour respot
{
while (respotted == false)
{
switch (ballNumber)
{
case 0:
ballID = cueBallGeom;
break;
case 1:
ballID = colourBallGeom[0];
break;
case 2:
ballID = colourBallGeom[1];
break;
case 3:
ballID = colourBallGeom[2];
break;
case 4:
ballID = colourBallGeom[3];
break;
}

freeCount = 0;
dGeomID ballID;
dGeomID testBall;
//get Geom of the ball being respotted
switch (ballNumber)
{
case 0:
ballID = cueBallGeom;
break;
case 1:
ballID = colourBallGeom[0];
break;
case 2:
ballID = colourBallGeom[1];
break;
case 3:
ballID = colourBallGeom[2];
break;
case 4:
ballID = colourBallGeom[3];
break;
}
}
case 5:
    ballID = colourBallGeom[4];
    break;
case 6:
    ballID = colourBallGeom[5];
    break;
default:
    break;
}

//checks against all other balls on the table
for (int i = 0; i < 22; i++)
{
    if (offTable[i] != 1)
    {
        testBall = 0;
        if ((i == 0) && (ballNumber != i))
        {
            testBall = cueBallGeom;
        }
        else if ((i > 15) && (ballNumber + 15 != i))
        {
            testBall = colourBallGeom[i-16];
        }
        else if ((i > 0) && (i < 16))
        {
            testBall = redBallGeom[i-1];
        }
        //check for potential collision against ball to be respotted
        if ((ballID != 0) && (testBall != 0))
        {
            dContact testContact[N];
            int n = dCollide(ballID, testBall, N, &testContact[0].geom, sizeof(dContactGeom));
            //if a contact point is returned then a collision takes place
            if (n == 0)
            {
                freeCount++;
            }
        }
    }
}

//if spot is free from all 21 balls return true
if (freeCount == ballsInPlay-1) { return true; } else { return false; }

/**
calculateScore: calculates the score after each shot
**/
int CGFfOpenGL::calculateScore(int cueBallPotted, int redPotted, int colourPotted)
{
    int scored = 0;
    int foul = 0;

    //work out what was hit first and if that is a foul
    if (ballHit == 0) //if no ball was hit
    {
        foul = foul + foulIndex(ballToPot);
        resetBallToPot();
    }
    else if ((ballHit == ballToPot) || ((ballHit > 1) && (ballToPot == 8))) //if correct ball was hit first,
    //just deal with illegal pots
    {
        if (cueBallPotted == 1) //if cueball has been potted
        {
            foul = foul + foulIndex(ballHit);
            resetBallToPot();
        }
        if ((colourPotted > 0) && (colourPotted < 8))
        {
            respot(colourPotted-1);
        }
        else if (ballToPot == 1) //red ball was the target ball
        {
            if (colourPotted != 0) //was a colour was also potted
            {
                foul = foul + foulIndex(colourPotted);
            }
            else if (redPotted != 0)
            {
                //add on number of red balls potted
                scored = scored + redPotted;
            }
            //set the colour to go for
            ballToPot = 8;
        }
        else if (redPotted == 1)
        { //add on number of red balls potted
            scored = scored + redPotted;
        }
        else if (ballToPot == 0)
        { //add on number of red balls potted
            scored = scored + redPotted;
        }
        //set the colour to go for
        ballToPot = 8;
    }
    return scored;
}
else  //colour was target
{
    if ((BallToPot == 8) && (redPotted != 0)) //was a red potted
    {
        foul = foul + foulIndex(ballToPot);
        resetBallToPot();
    }
    else if (colourPotted == 0)
    {
        resetBallToPot();
    }
    else if (colourPotted != 0)
    {
        foul = foul + foulIndex(colourPotted);
        resetBallToPot();
    }
    else //if not calculate score
    {
        resetBallToPot();
    }
}

else if (colourPotted == 0)
{
    resetBallToPot();
}
else //illegal ball hit first
{
    if (colourPotted != 0)
    {
        foul = foul + foulIndex(colourPotted);
    }
    else //illegal ball hit first
    {
        if (colourPotted != 0)
        {
            foul = foul + foulIndex(colourPotted);
        }
        else //illegal ball hit first
        {
            foul = foul + foulIndex(ballHit);
        }
        if (foulIndex(ballHit) > foul)
        {
            foul = foulIndex(ballHit);
        }
        if ((colourPotted > 0) && (colourPotted < 8))
        {
            respot(colourPotted-1);
        }
    }
    else //illegal ball hit first
    {
        foulState = 0;
        if (scored != 0)
        {
            return 1;
        }
        else return 0;
    }
}

/**
 * resetBallToPot: sets target ball
 **/ 
void CGfxOpenGL::resetBallToPot()
{
    if (ballsInPlay > 7)
    {
        ballToPot = 1;
    }
    else if (ballToPot == 8)
    {
        ballToPot = 2;
        onColours = 1;
    }
}
}
```c
/**
 * foulIndex: finds foul dependent on index value received
 */
int CGfxOpenGL::foulIndex(int index)
{
    // test for all 4 point foul conditions, first
    if ((index == 0) ||  // just the cueball potted -> 4 point foul
        (index == 1) ||  // no ball hit -> 4 point foul
        (index == 2) ||  // yellow ball foul -> 4 point foul
        (index == 3) ||  // green ball foul -> 4 point foul
        (index == 4) ||  // brown ball foul -> 4 point foul
        (index == 8))   // two of yellow/green/brown potted -> 4 point foul
    {
        return 4;
    }
    else if ((index == 5) || // blue ball foul -> 5 point foul
              (index == 9))
    {
        return 5;
    }
    else if ((index == 6) || // pink ball foul -> 6 point foul
             (index == 10))
    {
        return 6;
    }
    else if ((index == 7) || // black ball foul -> 7 point foul
              (index == 11))
    {
        return 7;
    }
    return 0;
}

/**
 * setCamera: Sets camera to relevant position
 */
CCamera CGfxOpenGL::SetCamera(CCamera Camera, float rotateX, int viewIndex, int ballState)
{
    // clear screen and depth buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // load the identity matrix (clear to default position and orientation)
    glLoadIdentity();
    if (viewIndex == 0)
    {
        const dReal * cueBallReal = dBodyGetPosition(cueBall);
        Camera.xView = cueBallReal[0];
        Camera.yView = cueBallReal[1];
        Camera.zView = cueBallReal[2];
        glColor3f(1.0, 0.0, 0.0);
        glLoadIdentity();
        glLoadIdentity();
    }
    else if (viewIndex == 1)
    {
        gluLookAt(0, 500, 0, 0, 0, 0, 0, 0, -1);
    }
    return Camera;
}

/**
 * moveCueball: Controls cue ball movement checking that ball doesn't collide
 * with other balls when being moved
 */
void CGfxOpenGL::moveCueball(char * dir)
{
    if (cueBallPotted == 1)
    {
        dVector3 tempPos;
        tempPos[0] = ballPos[0][0];
        tempPos[1] = ballPos[0][1];
        tempPos[2] = ballPos[0][2];
        dVector3 newPos;
        newPos[0] = ballPos[0][0];
        newPos[1] = ballPos[0][1];
        newPos[2] = ballPos[0][2];
        // checks movement direction
        if (!strcmp(dir, "D"))
        {
            dBodySetPosition (cueBall,tempPos[0],tempPos[1],newPos[2]);
            ballPos[0][2] = newPos[2];
        }
        else if (!strcmp(dir, "U"))
        {
            dBodySetPosition (cueBall,tempPos[0],tempPos[1],newPos[2]);
        }
    }
}
ballPos[0][2] = newPos[2];
}
else if (!strcmp(dir, "L")
{
    newPos[0] = tempPos[0] + 0.25;
    dBodySetPosition (cueBall,newPos[0],tempPos[1],tempPos[2]);
    ballPos[0][0] = newPos[0];
}
else if (!strcmp(dir, "R")
{
    newPos[0] = tempPos[0] - 0.25;
    dBodySetPosition (cueBall,newPos[0],tempPos[1],tempPos[2]);
    ballPos[0][0] = newPos[0];
}
(newPos[0] > 30))
{
    dBodySetPosition (cueBall,tempPos[0],tempPos[1],tempPos[2]);
    ballPos[0][0] = tempPos[0];
    ballPos[0][2] = tempPos[2];
}
}

/**
Render: Controls all main functionality
increments world each time called
and checks ball state
**/
int CGfxOpenGL::Render(CCamera Camera, float rotateX, int ballState)
{
    SetupLighting();
    DrawTable();
    DrawBalls();
    dSpaceCollide (space,0,&nearCallback);
    dWorldStep (world,0.05);
    //if cueBall hit
    if (ballState == 1) (
        applyFriction();
        ballState = getBallState();
        if (ballState == 0) 
        { //reset values of potted balls
            cueBallPotted = 0;
            colourPotted = 0;
            redPotted = 0;
            playerOnBreak = 0;
            foulState = 0;
            //check status at end of turn // apply rules
            cueBallPotted = checkPotted(1);
            //store number of reds potted
            redPotted = checkPotted(2);
            //check for colours potted
            colourPotted = checkPotted(3);
            playerOnBreak = calculateScore(cueBallPotted, redPotted, colourPotted);
        }
        if (playerOnBreak == 0)
        { //reset ballHit to 0 before next shot
            ballHit = 0;
        }
        dJointGroupEmpty(contactgroup);
    return ballState;
}
/**
 * RenderScores: draws the score display window
 */
void CGfxOpenGL::RenderScores(int width, int height, float powerPercent)
{
    // SCORE VIEWPORT
    glViewport (0, 0, width, (height - (height/1.05)));
    glMatrixMode (GL_PROJECTION);    // Select The Projection Matrix
    glLoadIdentity ();       // Reset The Projection Matrix
    gluPerspective( 45.0, (GLfloat)(width)/(GLfloat)(height), 0.1f, 500.0 );
    glMatrixMode(GL_MODELVIEW);    // set modelview matrix
    glLoadIdentity();
    glClear (GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glClear (GL_DEPTH_BUFFER_BIT);
    glTranslatef(0.0f,0.0f,-7.0f);
    glColor3f(1.0f, 1.0f, 1.0f);
    glBegin(GL_QUADS);
    glVertex3f( width,  5.0f, 0.0f);
    glVertex3f(-width,  5.0f, 0.0f);
    glVertex3f(-width, -5.0f, 0.0f);
    glVertex3f( width, -5.0f, 0.0f);
    glEnd();
    // checks whose turn it is and highlights font accordingly
    if (player == 1)
    {
        glColor3f(1.0f, 0.0f, 0.0f);
        glRasterPos2f(-3.7, -1);
        glPrint("Player 1  -  %d (Break %d)", playerOneScore, playerOneBreak); // Print GL Text To The Screen
    } else
    {
        glColor3f(0.5f, 0.5f, 0.5f);
        glRasterPos2f(-3.7, -1);
        glPrint("Player 1  -  %d", playerOneScore); // Print GL Text To The Screen
    }
    if (player == 2)
    {
        glColor3f(1.0f, 0.0f, 0.0f);
        glRasterPos2f(1.5, -1);
        glPrint("(Break %d) %d  -  Player 2", playerTwoBreak, playerTwoScore); // Print GL Text To The Screen
    } else
    {
        glColor3f(0.5f, 0.5f, 0.5f);
        glRasterPos2f(2.4, -1);
        glPrint(" %d  -  Player 2", playerTwoScore); // Print GL Text To The Screen
    }
    // outputs power indicator text
    glColor3f(0.0f, 0.0f, 0.0f);
    glRasterPos2f(-0.8, -1);
    glPrint("%7.2f%% Power", powerPercent); // Print GL Text To The Screen
}

/**
 * RenderAimLine: draws aim line viewport on top of the main viewport
 */
void CGfxOpenGL::RenderAimLine(int width, int height, int ballState, int viewIndex)
{
    if ((ballState == 0) && (viewIndex == 0))
    {
        //AIM LINE VIEWPORT
        glViewport (0, 0, width, height);
        glMatrixMode (GL_PROJECTION);    // Select The Projection Matrix
        glLoadIdentity ();       // Reset The Projection Matrix
        gluPerspective( 45.0, (GLfloat)(width)/(GLfloat)(height), 0.1f, 500.0 );
        glMatrixMode(GL_MODELVIEW);    // set modelview matrix
        glLoadIdentity();
        glClear (GL_DEPTH_BUFFER_BIT);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glClear (GL_DEPTH_BUFFER_BIT);
        glTranslatef(0.0f,0.0f,-7.0f);
        glColor3f(1.0f, 1.0f, 1.0f);
        glEnable(GL_LINE_STIPPLE);
        short stipplePattern = 0xAAAA;
        glLineStipple(4, stipplePattern);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glClear (GL_COLOR_BUFFER_BIT);
        glTranslatef(0.0f,0.0f,0.0f);
        glLineStipple(4, stipplePattern);
        glBegin(GL_LINES);
        glVertex3f(0.0f,0.0f,0.0f);
        glVertex3f(0.0f,0.0f,2.0f);
        glEnd();
        // if a foul has been committed
        if (foulState == 1)
        {
            glColor3f(1.0f, 1.0f, 1.0f);
            glRasterPos2f(-0.5, -1.5);
            glPrint("Foul!");
        }
    }
}
gIPrint("FOUL SHOT");  // Print GL Text To The Screen

}