Gaze and Voice Based Game Interaction

by

Jonathan O'Donovan, B.Sc.

Dissertation

Presented to the

University of Dublin, Trinity College

in partial fulfillment

of the requirements

for the Degree of

Master of Computer Science in Interactive Entertainment Technology

University of Dublin, Trinity College

September 2009

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Jonathan O'Donovan

September 9, 2009

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Jonathan O'Donovan

September 9, 2009

Acknowledgments

The author would like to thank Dr. Veronica Sundstedt for all her guidance and advice during the project, Paul Masterson for his help in all matters hardware related both during the project itself and indeed throughout the duration of the course. The author would also like to thank Jon Ward for kindly lending a Tobii T60 eye tracker for use in the project.

Jonathan O'Donovan

University of Dublin, Trinity College
September 2009

Gaze and Voice Based Game Interaction

Jonathan O'Donovan
University of Dublin, Trinity College, 2009

Supervisor: Veronica Sundstedt

Modern eye tracking technology allows an observer's gaze to be determined in real-time by measuring their eye movements. Recent studies have examined the viability of using gaze data as a means of controlling computer games. This dissertation investigates the combination of gaze and voice recognition as a means of hands-free interaction in 3D virtual environments. A game evaluation framework is implemented controllable by input from gaze and voice as well as mouse and keyboard. This framework is evaluated both using quantitative measures and subjective responses from participant user trials. The main result indicates that, although game performance was significantly worse, participants reported a higher level of enjoyment and immersion when playing using gaze and voice.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Eye tracking is a process which measures the motion of the eye. This gaze data can be used to determine an observer's *point-of-regard* (POR) in real-time. Combining gaze data with voice recognition offers a hands-free alternative to conventional video game interaction. The work presented in this dissertation hopes to show that gaze and voice controlled game interaction is a viable alternative to mouse and keyboard, which can augment game play, providing an enhanced, more immersive user experience.

1.1 Motivation

Recent innovations such as Nintendo's Wii have illustrated how popular alternative means of computer game interaction can be. The Wii sold over twice as many units as the PlayStation 3 from when they were released to the middle of 2007 [1]. As eye trackers become cheaper and less intrusive to the user the technology could well be integrated into the next generation of games. It is important therefore to ascertain its viability as an input modality and see how it can be used to enhance gamer experience.

Gaze based interaction is not without its problems. It tends to suffer from the "Midas touch" problem. This is where everywhere you look, another command is activated; you cannot look anywhere without issuing a command [2]. To combat this gaze is often used in conjunction with another input mechanism such as a mouse click. The intention of this work is to show that the "Midas touch" problem can be overcome by combining voice recognition with gaze, to achieve a completely hands-free method

of game interaction.

Alternative means of interaction in games are especially important for disabled users for whom traditional techniques, such as mouse and keyboard, may not be feasible. Given that gaze and voice is entirely hands-free it presents a real opportunity for disabled users to interact fully with computer games.

1.2 Objectives

The main objective of this project is to develop a video game using eye tracking and voice recognition as a means of controlling it. Eye movement data will be obtained and processed in real-time using a Tobii T60 eye tracker. Microsoft Speech SDK 5.1 will be used to process and interpret voice commands.

The game will be purpose built for evaluation tests in a controlled manner. It will facilitate the saving of all relevant game data for later analysis. The game is to be controllable by mouse/keyboard, gaze/voice, mouse/voice and gaze/keyboard. This will allow comparisons to be drawn between the different input types.

A user study will be conducted to evaluate how suitable gaze and voice is as a means of video game control. Being controllable by mouse and keyboard as well as voice and gaze will facilitate direct comparisons between the two methods of interaction. In addition to the saved game data questionnaires will be given to participants to ascertain subjective data, such as how effective gaze and voice are perceived to be and how immersive and entertaining the experience was.

1.3 Dissertation Layout

Chapter 2 reviews the state of the art with regard to the use of gaze and voice recognition in video games. The design and implementation of the gaze and voice based computer game are described in Chapters 3 and 4. Chapter 5 focuses on the evaluation of the results gathered in a user trial of the game. Finally in Chapter 6 conclusions are drawn and related future work is suggested.

Chapter 2

Background & Related Work

This chapter provides an overview of human vision, eye movements and how they are studied using eye tracking technologies in Sections 2.1 - 2.3. Section 2.3 explores the hierarchy of eye tracking systems with particular emphasis on interactive systems. Sections 2.4 - 2.6 review the state of the art with regard to the use of voice recognition and gaze in video games.

2.1 Human Vision

Human vision is a gradual process where the field of view is inspected little by little through brief fixations over small regions of interest. This allows for perception of detail through the central region of vision, also referred to as the foveal region. Central foveal vision allows for fine grained scrutiny of only a tiny proportion of the entire visual field. For example only 3% of the size of a 21" computer monitor is seen in detail when viewed at a 60 cm distance [3]. Peripheral vision is not good at determining fine detail. Figure 2.1 shows an example of the Hermann grid illusion, an optical illusion reported by Hermann in 1870 [4]. It is a visual example of the difference between foveal and peripheral vision. White dots are perceived along the lines crossing the fovea while black dots appear along the lines in the periphery [3].

Figure 2.2 shows how the visual axis runs from the midpoint of the visual field to centre of the fovea. At the back of eye is the retina. The retina contains light sensitive receptors which are classified into rods and cones. In all there are approximately 120

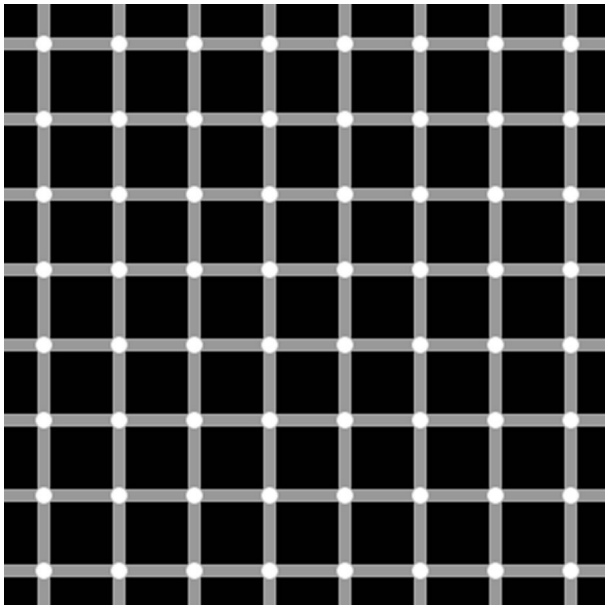

Figure 2.1: Hermann grid illusion. Reprinted from [5].

million rods and 7 million cones [3]. Cones respond better in daylight and rods in darker conditions such as at night [3].

How a viewed object subtends at the eye can be described in terms of degrees of visual angle as shown in Figure 2.2. The degree of visual angle is defined as

$$A = 2arctan\frac{S}{2D'} \tag{2.1}$$

where D is the distance to the object being viewed and S the size of the object [3]. The fovea contains 147,000 cones/mm^2 and a slightly smaller number of rods. At about 10° the number of cones drops sharply to less than 20,000 cones/mm^2 and at 30° the number of rods in the periphery drops to about 100,000 rods/mm^2 [7].

2.2 Eye Movements

Repositioning of the fovea can be attributed to combinations of five basic types of eye movement: *saccadic*, *smooth pursuit*, *vergence*, *vestibular* and *physiological nystagmus* [8]. Positional eye movements are important when using eye tracking as a means of computer interaction since they indicate the position of the fovea and therefore the

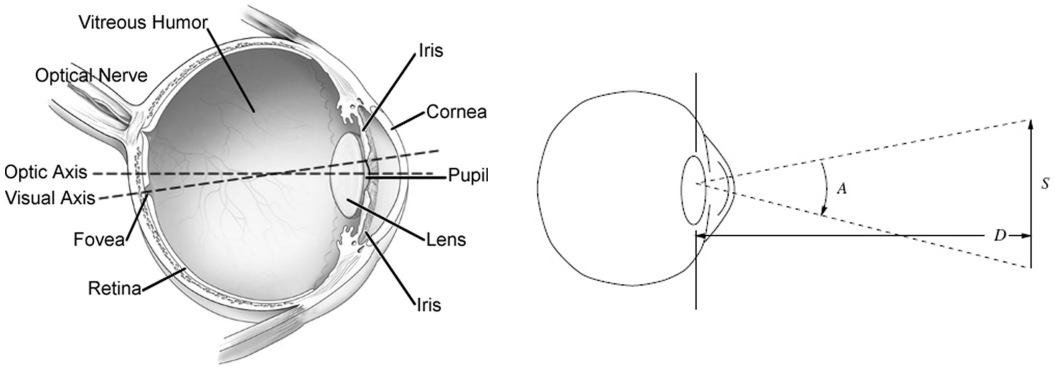

Figure 2.2: The structure of the human eye showing the visual axis (left) and (right) the degree of visual angle. Adapted from [6] and reprinted from [3].

point-of-regard (POR) of the viewer.

When visual attention is directed to a new area of interest fast eye movements known as saccades reposition the fovea. Saccades can be both voluntary and involuntary and range in duration from 10 - 100 ms [3] rendering the executor effectively blind during the transition [9]. Saccades are considered manifestations of the desire to voluntarily change the focus of attention.

Repositioning of the fovea is carried out by pursuit movements when visually tracking a moving object. The eyes are capable of matching the velocity of the moving target depending on the range of target motion. Pursuits correspond to the desire to maintain ones gaze on an object in smooth motion.

Fixations are eye movements that stabilize the fovea over a stationary object of interest. They naturally correspond to the desire to maintain ones gaze on an object of interest. Fixations are characterized by the miniature eye movements: *tremor*, *drift* and *microsaccades* [10]. Counterintuitive as it may seem if an image were to be artificially stabilized on the retina, vision would fade away within about a second and the scene would become blank [11]. Typically these microsaccades are no larger than 5° visual angle [12]. Fixations duration range between 150 - 600 ms. It is estimated that 90% of viewing time is devoted to these tiny movements [13].

Nystagmus eye movements are a combination of eye movements. Vestibular nystagmus is a smooth pursuit movement interspersed with saccades. They usually occur to compensate for the movement of the head [3]. Optokinetic nystagmus is a similar type of eye movement compensating for movement of a visual target [3].

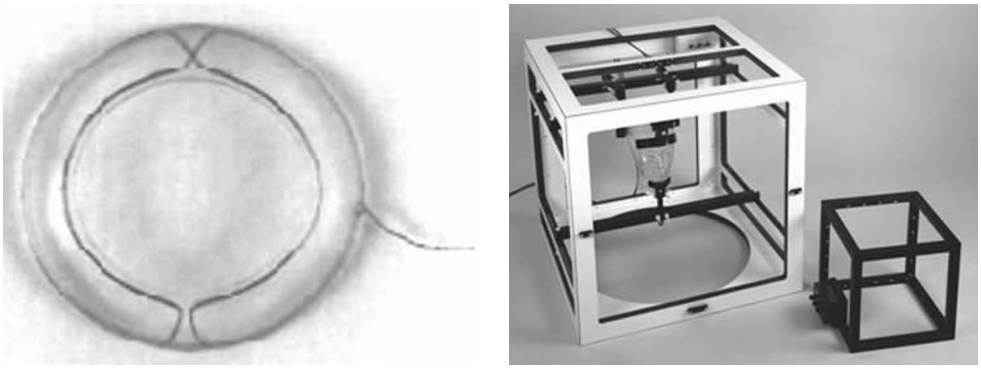

Figure 2.3: Scleral contact lens/search coil eye tracker, (left) contact lens with embedded search coil and (right) electromagnetic field frames. Reprinted from [14] and [3]

Other eye movements relating to pupil dilation and lens focusing include *vergence*, *adaptation* and *accommodation*. However it is only really the three types of movements fixations, saccades and smooth pursuits which need be modelled to gain insight into visual attention. These movements provide evidence of voluntary, overt visual attention [3].

2.3 Eye Tracking Technology

Eye trackers are devices used to measure the movements of the eye by measuring either the motion of the eye relative to the head or the POR. One of the first measurements of eye movement was made using corneal reflection in 1901 [3]. This section provides an overview of eye tracking technology in use today.

2.3.1 Eye Tracking Categorisations

The four main categories of eye trackers in use today, are *Scleral contact lens/search coil*, *Electro-OculoGraphy* (EOG), *Photo-OculoGraphy* (POG) or *Video-OculoGraphy* (VOG) and *video-based combined pupil and corneal reflection* [3].

Figure 2.3 shows the Scleral contact lens/search coil technique of eye tracking which came to prominence in the 1950s. It involves placing a contact lens on the eye. The contact lens being attached to coils or mirrors used to record eye movements [3]. This technique is very precise but suffers from two main drawbacks. First it is very invasive

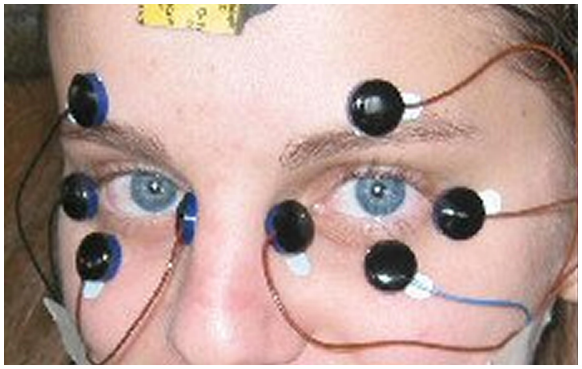

Figure 2.4: EOG eye movement measurement. Reprinted from [15].

causing discomfort for the user. Second it measures the eye position relative to the head and meaning unless used in conjunction with a head tracker is not suitable for finding the POR.

EOG uses measurements of the skin's electric potential differences, gathered through the use of electrodes placed around the eye, as shown in Figure 2.4. This method also only measures eye positions relative to the head and is therefore not suitable for obtaining the POR [3].

POG and VOG categories of eye tracking groups together a variety of eye movement recording techniques. These techniques involve measurement of features of the eye under rotation or translation, such as the shape of the pupil, the position of the iris-sclera boundary and corneal reflections [3]. Although the measurements are different in approach they are grouped together since they normally do not provide POR.

The above categories of eye movement measurements do not provide POR. To obtain the POR the head can be fixed or some method of distinguishing head movements from eye rotations used [3]. To differentiate between the two types of movements, features of the eye can be recorded and analysed. Features which can achieve this include the pupil centre and corneal reflection. Corneal reflection is recorded using directed light sources, usually infra-red. Video-based combined pupil and corneal reflection trackers compute the POR in real-time using cameras and image processing hardware [3]. They may be a table or head mounted apparatus as shown in Figure 2.5. When these eye trackers are correctly calibrated they are capable of measuring an observer's POR on suitably positioned, perpendicularly planar surfaces [3]. They are the most commonly used eye trackers in use today.

Figure 2.5: Left shows a Tobii X120 table mounted eye tracker. An EyeLink II head mounted eye tracker is shown right. Reprinted from [16] and [17].

2.3.2 Calibration

Video-based combined pupil and corneal reflection eye trackers require calibration. This involves presenting a sequence of on screen visual stimuli, at various extents of the viewing region. By measuring the pupil position and corneal reflection the POR can be calculated at these extents. These values can then be interpolated across the viewing region [3]. Modern eye trackers often provide inbuilt calibration procedures. The visual stimuli used in eye trackers manufactured by Tobii, for example are a sequence of circles which appear at various positions on screen. The number of stimuli can be set between 2, 5, or 9 points.

The primary purpose of calibration is to gather a sufficiently large range of co-ordinates to allow interpolation of the viewer's POR between extrema points (e.g. upper-left, lower-right, upper-right and lower-left) [3]. It is also used to help the eye tracker from losing the pupil or corneal reflection when faced with artifacts such as eyelashes, glasses or contact lens. This is done by setting thresholds for pupil and corneal reflection recognition [3].

2.3.3 Fixation Detection & Eye Movement Analysis

Eye movement analysis can be used to gain insight into the viewer's visual attentive behaviour [3]. As previously stated in Section 2.2, fixations tend to indicate where a viewer's attention is directed and saccades tend to indicate a viewer's wish to change the area of their focus of attention. Modeling eye movements as signals and approximating

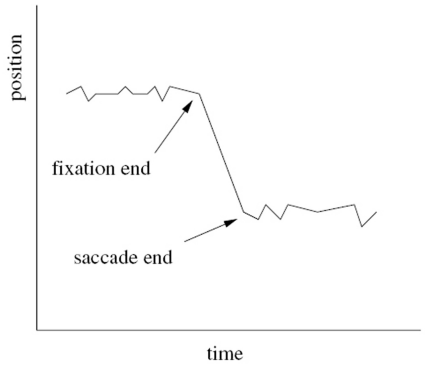

Figure 2.6: A hypothetical eye movement signal. Reprinted from [3].

them to linear filters provides a relatively easy way of analysing them [3].

Figure 2.6 shows a hypothetical eye movement signal. Two methods are commonly used to detect saccades and fixations from a given signal. Summation involves averaging the signal over time, when little or no variance is recorded then the signal can be deemed a fixation. A second method uses differentiation where successive signals are subtracted to estimate velocity. This method explicitly detects saccades and implicitly detects fixations, that is, where there is little or no difference between successive signals [3].

Eye movements signals are prone to noise due to blinking and the inherent instability of the eye [3]. This noise needs to be removed prior to any analysis. Usually this can be done by eliminating any signals outside a given rectangular range. The rectangular area selected should also exclude areas outside the "effective operating range" of the eye tracker itself. This will also remove noise due to blinks, since most eye trackers return coordinates of (0,0) when a signal is lost [3].

2.3.4 Eye Tracking Systems

Figure 2.7 shows the hierarchy of eye tracking systems. Diagnostic systems are primarily used to provide evidence of user's attention processes. This dissertation is primarily concerned with interactive eye tracking systems. There are two main types of interactive applications using eye tracking technology: *selective* and *gaze-contingent*. Selective uses an eye tracker as an input device, in a similar way to a mouse. Gaze-contingent applications are a type of display system where the presented information is manipulated in some way to match the processing capability of the human visual system. This

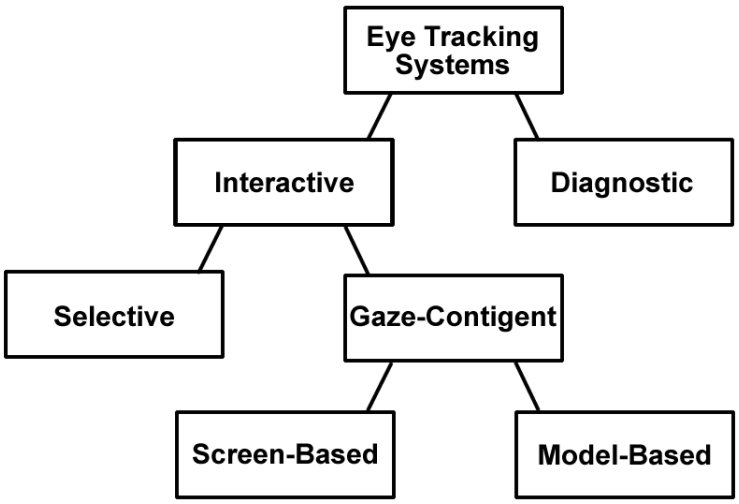

Figure 2.7: The hierarchy of eye tracking systems. Adapted from [3].

often matches foveal/peripheral perception in real-time.

Jacob [2] presented one of the first papers looking into the feasibility of gaze based selective interaction and identified the "Midas touch" problem, discussed in Section 1.1. Jacob suggested dwell time as a selection mechanism to over come the problem. In the same year Starker and Bolt [18] introduced one of the first documented prototypes with computerised real-time eye tracking and intentionally constructed storytelling. It featured a gaze responsive self disclosing display. A rotating planet was displayed with various features including volcanoes, staircases and flowers, as shown in Figure 2.8. When the observers gaze dwelt on these objects long enough, the system gave more information about the object of interest using synthesized speech. While this may not be considered a game, many games have similar features, such as exploring ones surroundings and interacting with objects and characters. Section 2.5 describes gaze interaction in video games in more detail.

Gaze-contingent displays (GCDs) attempt to balance the amount of information displayed against the visual information processing capacity of the observer [20]. Hillaire *et al.* [19] developed an algorithm to simulate depth-of-field blur for first-person navigation in virtual environments, as shown in Figure 2.8. Using an eye-tracking system, they analysed users focus point during navigation in order to set the parameters of the algorithm. The results achieved suggest that the blur effects could improve the sense of realism experienced by players. O'Sullivan and Dingliana [21] and O'Sullivan *et al.* [22]

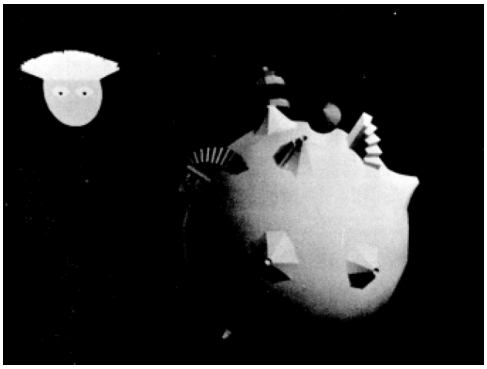
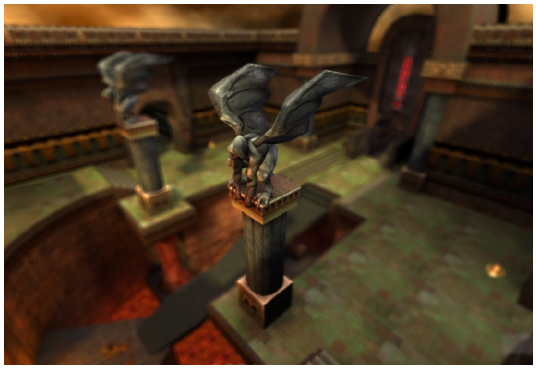

Figure 2.8: Left shows Starker and Bolt's 3D graphics display. Right shows a Quake III video game with blur effects implemented. Reprinted from [18] and [19].

take another approach, instead of resolution degradation of peripheral objects, collision handling outside the foveal region of interest was degraded. In other words object collisions were allocated more processing time when being examined more closely than collisions occurring in peripheral vision. This resulted in in more believable, physically accurate collision responses in the regions of interest. The collisions occurring in the periphery were given less processing time, meaning they would be less accurate.

2.4 Voice Recognition in Gaming

This section gives an overview of how voice recognition works and examines some examples of its use in gaming. There are two different categories of speech recognition technologies, speaker dependent and speaker independent [23]. Speaker-dependent requires each user to go through a process of training the engine to recognise his/her voice. Speech independent recognition avoids this by training with a collection of speakers in the development phase. The process of speech recognition can be divided into the following steps [24]:

- **Recognition grammar:** specifies and defines the speech input and its pattern to the speech recognition engine.
- **Phoneme identification:** Incoming audio signals are analysed and compared to language phonemes.
- **Word identification:** The resulting output of phonemes are compared to words in the recognition grammar.

Console Vendor	Game	Publisher	Developer	**Speech Technology**
Microsoft: Xbox	Ghost Recon 2	Ubisoft	Red Storm	Fonix
	Rainbow Six 3	Ubisoft	Red Storm	Fonix
	Rainbow Six 3: Black Arrow	Ubisoft	Red Storm	Fonix
	SWAT: Global Strike Team	Vivendi Universal/Sierra	Argonaut	Fonix
Nintendo: GameCube	Mario Party 6	Nintendo	Hudson Soft	ScanSoft
Sega: Dreamcast	Seaman (Japan)	Sega	Sega	ScanSoft
Sony: PlayStation 2	Deka Voice (Japan)	Sony Computer	Sony Computer	ScanSoft
	Ghost Recon 2	Ubisoft	Red Storm	Fonix
	Ghost Recon 2: Jungle Storm	Ubisoft	Red Storm	ScanSoft
	LifeLine	Konami Digital	Konami Digital	ScanSoft
	Operators Side (Japan)	Konami Digital	Konami Digital	ScanSoft
	Rainbow Six 3	Ubisoft	Red Storm	ScanSoft
	SOCOM: U.S. Navy Seals	Sony Computer	Sony Computer	ScanSoft
	SOCOM II: U.S. Navy Seals	Sony Computer	Sony Computer	ScanSoft
	SWAT: Global Strike Team	Vivendi Universal/Sierra	Argonaut	ScanSoft

Table 2.1: Video games with speech recognition. Recreated from [25].

- **Output:** The output produced is the best guess the engine can construe from the user's input.

Table 2.1 lists games which have used voice recognition. Some of the main issues with the use of speech recognition in gaming include [26]:

- Work best in quite environments (not usually a natural gaming environment).
- Inconsistency, can work well with one person and badly with the next.
- The larger the vocabulary the easier it is to confuse the engine (generating false positive recognitions).
- Long words are easier to recognise but short words are more common in language.
- Tends to be computational expensive, making results in real-time problematical.

Mehdi *et al.* [23] used natural language speech recognition to instruct virtual characters in a 3D environment. Speech synthesis was also used to respond to the user. So if a user instruction was incomplete for example "put the cup", the virtual character would ask "Where do you want to put the cup?" While this interaction is interesting no results where provided as to how accurate the recognition was or as to how quickly the virtual character reacted to voice input.

Hämäläinen *et al.* [27] developed a musical edutainment game in which the game character was controlled using pitch. The idea was that the player had to sing in tune to move a character up and down a path to its destination. This novel approach had

some issues. The pitch detection sometimes reacted to the background music when the user was not singing. Acoustic echo cancellation (AEC) methods were suggested as a possible solution to remove the music from the input signal. This could be used in other games where voice recognition is used as an input mechanism. This would allow game music to be played creating a more normal game environment. However background noise would still be an issue.

2.5 Gaze in Gaming

When looking at eye tracking as a means of computer game interaction it is important to look at different genres and the challenges they present. Isokoski *et al.* [28] discuss this in some detail, a summary of which is presented in Section 2.5.1. Section 2.5.2 examines in more detail some previous work looking at gaze input in gaming.

2.5.1 Game Genre Implications for Gaze Input

For board games, puzzles and card games the main issues lie with updating games expecting direct input from keyboard or mouse. Card games also tend to use bitmap graphics, this makes it difficult to resize the interface for adapted use with eye trackers.

Shoot-em ups, or scrolling shooters, use two degrees of freedom in addition to a trigger. Typically these involve evading enemy missiles/projectiles while shooting as many targets as possible. The problem for eye tracking is that the player often needs to look at a target while simultaneously staying out of the line of fire. So gaze position cannot be directly linked to avatar position.

Beat-em up games typically involve sequences of timed operations in order to perform a particular move. This presents many challenges to eye trackers. Namely how to associate the many varied moves involved with gaze.

The same may be said for the First Person Shooter (FPS) genre of game. While these games are presented in first person perspective which typically lends itself well to eye tracking there are usually also a number of controls to be invoked to fire, reload and interact with objects in the game. Another issue is that of accuracy. Eye trackers are not competitive with the sub-pixel positioning accuracy of modern mice. Racing games present similar problems to FPS genre games. It may be difficult to associate

Game genre	Indicators for eye tracker use						
	Positive		Negative				
	One player mode	Turn-based gameplay	Online multiplayer	Online real-time multiplayer	Continuous position control	Dissociation of focus of attention and control	Large number of commands
Board games and puzzles	x	x	x				
Card games	x	x	x				
Shoot-em up	x				x	x	
Beat-em up	x				x	x	x
First person shooters	x			x	x		
Flight simulators	x			x	x	x	x
3rd person action and adventure	x						x
Level jumping (platform)	x				x	x	
Turn-based Strategy	x	x	x				x
Real-time strategy	x			x			x
Turn-based role playing games	x	x					x
Real-time role playing games	x			x			x
Racing	x			x	x		

Table 2.2: Positive and negative indicators for eye tracker compatibility of a game genre. Recreated from [28].

gaze directly to steering as it may be necessary to look at other parts of the screen without changing the current direction.

Flight simulators require observing of the surrounding environment while also manipulating the controls in order to fly the airplane. Again this is difficult to achieve using eye tracker input. Adventure games and level jumping platform games show the player in the third person as if the cameraman were following the game character. Again it may be difficult to invoke all the controls required to move the character to desired locations.

Turn-based strategy and turn-based role playing games should be fairly compatible with eye trackers provided the interface can be enlarged sufficiently to allow their use. However some elements of the interface may present problems for inputting data or commands. This could perhaps be overcome with the use of a menu system. Real-time strategy games are similar except that the game evolves in parallel with the players input, not in reaction to it. So eye tracker use may prove too slow to compete against the computer or indeed other online players not using eye tracking input. Table 2.2 shows a summary of positive and negative indicators for eye tracker compatibility for different genres.

2.5.2 Gaze Input In Gaming

The primary goal of Leyba and Malcolm's [29] work was to examine the performance variations of eye tracking versus mouse as an aiming device in a computer gaming environment. They hoped to show that although user accuracy would be less, task completion times would improve. A simple 3D eye tracking game utilising the Tobii ET-1750 eye tracker was developed. Twenty-five balls with random velocity vectors were randomly placed on the screen. The balls travelled according to a simple physics simulation. The aim of the game was to remove the balls by clicking on them. To overcome the "Midas touch" a conservative form of "Manual and Gaze Input Cascaded (MAGIC) pointing" [30] was used. MAGIC pointing "warps" the cursor to the general POR. The user can then make small adjustments to be directly on target with a mouse. Leyba and Malcolm's adapted method used the gaze point as the cursor position on screen when the mouse was clicked. They found that as expected mouse input was more accurate. However contrary to their hypothesis completion time also proved longer, requiring an average of 49.98 extra seconds to complete the objective. This was most likely due to problems with calibration and users tending to repeatedly click the mouse rather than aim accurately when using the mouse as the input.

Kenny *et al.* [31] created a diagnostic eye tracking system. A FPS game was created using mouse and keyboard as input. Using an EyeLink II head-mounted eye tracker the players' gaze was recorded while they played and later analysed. It was found that the cross hairs in a FPS game acts as a natural fixation point. 88% of fixations and 82% of game time took place in the 'near centre' of the screen (the inner 400 x 300 rectangle from the 800 x 600 resolution screen). Surprisingly only 2% of fixations occurred within regions where health, status messages and score appeared. Kenny *et al.* suggest that this information could be exploited to improve graphics rendering algorithms, rendering central regions to a higher quality than the periphery. However this observation may also prove useful for gaze based video game interaction. The areas in the periphery could be used to fire commands using fixation while not interfering with the normal FPS game play.

Jönsson's [32] thesis experimented with the FPS, Half Life, and the Shoot-em-up, Sacrifice. Open source versions of these games were adapted to accept gaze input from a Tobii eye tracker. Two demos were created using Sacrifice, one were aim was controlled

by mouse and one by gaze. In Half Life three demos were created, one were the weapon sight and field of view were controlled by mouse, one were weapon sight and field of view were controlled by gaze and one where weapon sight was controlled by eyes and field of view with mouse. Participants achieved higher scores with eye control when playing Sacrifice than without it. However performance in Half Life, which potentially had the more interesting input techniques was not reported.

Three different game genres were examined by Smith and Graham [33] the FPS, Quake 2, a Role Playing game, Neverwinter Nights and a Shoot-em-up, Lunar Command. Eye tracking was used to control orientation, communicate with the avatar and target moving targets in Quake 2, Neverwinter Nights and Lunar Command respectively. To overcome the "Midas touch" problem all gaze based interaction was done in conjunction with mouse and keyboard. For orientating the camera in the gaze controlled Quake 2, users looked at objects which in turn rotated the camera to centre on that point. All three genres reported lower performance where gaze was used as compared to mouse control. The greatest discrepancy being Lunar Command, users seemingly finding it difficult to "lead" missiles by looking into empty space in front of moving targets.

Isokoski and Martin [34] and Isokoski *et al.* [35] developed a FPS style game which decoupled aiming from viewing. The game used gaze for aim, mouse to control the camera and the keyboard to move the player around the game world. This was measured against mouse and keyboard and the Xbox controller. Results suggested that the gaze controlled version was at least as accurate as the other input methods. However one must be skeptical of the results given only one participant, closely involved with the project, was involved in the evaluation of the game.

Dorr *et al.* [36] adapted a game, *Breakout*, to allow for input from either mouse or gaze. *Breakout* is a game, similar to *Pong*, in which a paddle is moved horizontally to hit a ball that is reflected off the borders of the playing area. The ball dissolves *bricks* in the upper part of the game area upon contact. The objective of the game is to destroy all bricks. The game was adapted by connecting the horizontal movement of the paddle with the player's gaze. *Breakout* proved to be well suited to gaze control and participants in the study performed better using gaze input. While gaze data cannot compete with the accuracy of a mouse, eye movements are faster. This along with the simple one-dimensional movement required to keep the ball in play suggests why gaze

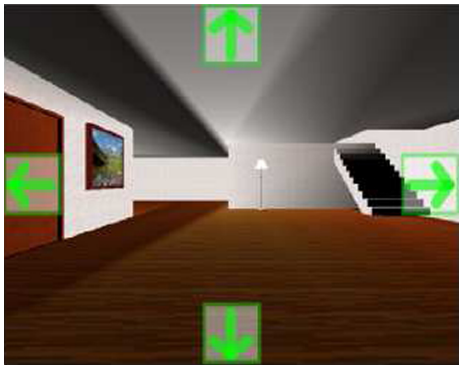

Figure 2.9: "Virtual Keys" used by Castellina and Corno. Reprinted from [37].

performed better than mouse input in this case. Also elements of the original open source game which did not work well with eye tracking input were removed entirely.

Castellina and Corno [37] developed some simple 3D game scenarios to test multimodal gaze interaction video games. Results showed that where gaze was the only input device it was as accurate as any of the other input devices used. While direct gaze control was not as fast as the other methods of interaction, this could be as a result of the use of dwell time to overcome the "Midas touch" problem and would perhaps not be an issue had another input modality, such as voice, been used instead. The direct gaze control used *VK*, or "Virtual Keys", four semitransparent buttons in the middle of screen edges. These allowed left and right camera rotation and forward backward navigation and were activated by dwell time as shown in Figure 2.9.

While accuracy and ease of use have been shown to be problematic when eye trackers have been used as input to video games very often subjective data gathered from participants have revealed some interesting opinions. For instance a large majority of participants in the study done by Smith and Graham[33] felt more immersed in the virtual environment when using the eye tracker. Conversely in Leyba and Malcolm's [29] work a majority of participants deemed mouse input as the preferred option over eye tracking. The analysis of the questionnaires taken in Castellina and Corno's [37] work showed that the *VK* method had been perceived as the most accurate and fastest control type despite the objective data gathered to the contrary. A majority of participants in Jönsson's usability study [32] rated Sacrifice and Half Life as more fun when controlled using gaze. It was also reported that the combination of weapon sight controlled by eyes and field of view with mouse felt more natural than mouse control.

Figure 2.10: Revenge of the Killer Penguins. Left shows a staring competition and right the catapult perspective. Reprinted from [38].

Perhaps the novelty factor and the immersion provided by gaze control gives players a more complete and better game experience.

2.6 Gaze & Voice in Gaming

Wilcox *et al.* [38] created the first and only game which used gaze and voice control. The game, a 3rd person adventure puzzle, could be controlled by both gaze and voice and by gaze alone. In the second modality blinks and winking were also used to activate commands. The work included some interesting features which utilised the characteristics of both gaze and voice input. For example a time lag in selecting items was used, which allowed time for voice commands to be recognised and processed. Two of the game features are shown in Figures 2.10. Unfortunately the work did not involve a user evaluation so it is difficult to judge the benefits or shortfalls of their approach.

Chapter 3

Design

This chapter examines the design issues involved in creating the gaze and voice game. Sections 3.1 and 3.2 describe the hardware and development tools used in the project. Section 3.3 discusses the game concept. Finally Section 3.4 gives an overview of the game framework design.

3.1 Hardware

A Tobii T60 eye tracker, shown in Figure 3.1, along with the Tobii Software Development Kit (SDK) was made available for use in the project from Acuity-ETS a reseller of Tobii eye trackers for the UK and Ireland. The T60 eye tracker is integrated into a 17" monitor and provides eye math models with advanced drift compensation, which allows large freedom of head movement [39]. The technical specifications for the T60 are shown in Table 3.1.

3.2 Development Tools

This section gives an overview of the development tools required to implement the gaze and voice game. Sections 3.2.1 and 3.2.2 describe the Tobii SDK and Microsoft Speech API respectively. Section 3.2.3 discusses which game development tools were investigated and which was finally settled upon. Section 3.2.4 gives an overview of the Microsoft Component Object Model (COM) and how .NET and COM communicate.

Figure 3.1: Tobii T60 eye tracker. Reprinted from [40].

Accuracy	0.5 degrees
Drift	0.3 degrees
Data rate	60 Hz
Freedom of head movement	44x22x30 cm
Binocular tracking	Yes
Bright/dark pupil tracking	Both - automatic optimization
TFT Display	17" TFT, 1280 x 1024 pixels
T/X firmware	Embedded
User camera	Built in

Table 3.1: The Tobii T60 technical specification. Data rate refers to the number of samples the eye tracker is capable of per second, 60Hz is generally considered sufficient to capture most eye movement. Recreated from [39].

3.2.1 Tobii SDK

The Tobii SDK enables development of applications for controlling and retrieving gaze data from Tobii eye trackers [41]. One of the interfaces Tobii SDK provides is the Tobii Eye Tracker Components API (TetComp). TetComp is a high level interface which provides ready-made COM objects for collection of gaze data and ready-to-use calibration, track status and calibration plotting tools. It hides much of the programming complexity of the lower level APIs.

The TetComp objects of particular interest for this dissertation are:

- **Tracking component:** `TetClient` COM object which allows the retrieval of gaze data.

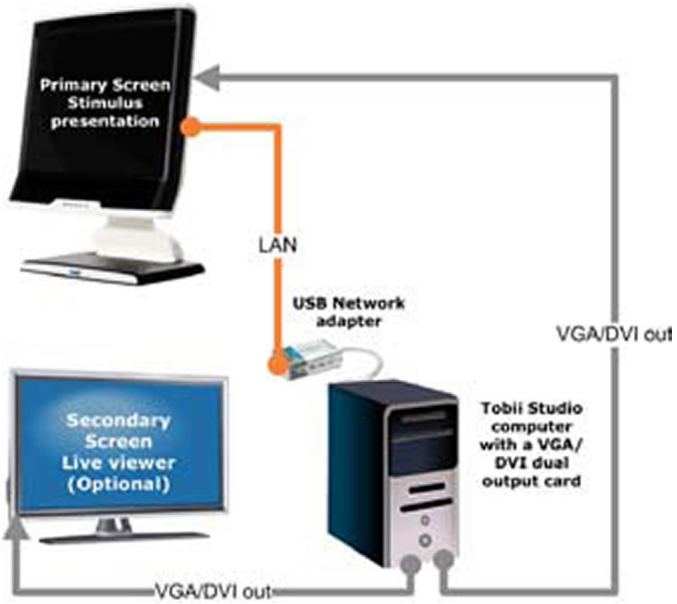

Figure 3.2: Tobii eye tracker setup. Reprinted from [42].

- **Calibration tools:** The `TetCalibManager`, `TetCalibProc` and `TetCalibPlot` COM objects provide tools for calibration.

Figure 3.2 shows a typical set up for a Tobii eye tracker. In the case of the integrated T60 a TCP/IP connection is established with the host workstation. The host will run the eye tracking application using TetComp to communicate with the eye tracker.

3.2.2 Microsoft Speech API

Microsoft Speech API (**SAPI**) provides a high-level interface between an application and speech engines, allowing for the use of speech recognition and synthesis within a Windows application [43]. SAPI communicates with applications by sending events using standard callback mechanisms, such as Window Message, callback procedure or Win32 Event. Figure 3.3 shows the different layers of SAPI.

Applications can control speech recognition using the `ISpRecoContext` COM interface. The interface is effectively a vehicle for receiving notifications for speech recognition events. The `SPEI_RECOGNITION` event is of particular interest in SAPI speech recognition applications. It is fired when the speech recognition engine returns a full recognition.

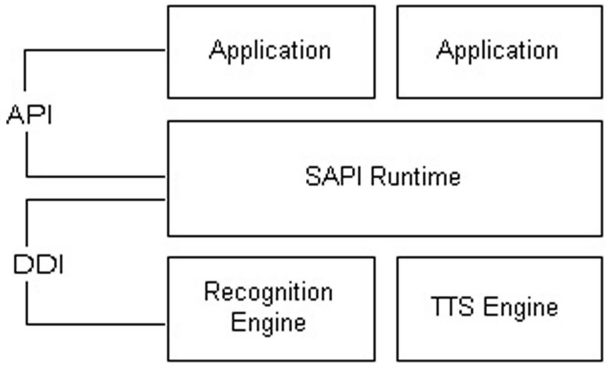

Figure 3.3: Different layers of SAPI. Reprinted from [43].

The `ISpRecoGrammar` SAPI interface enables the management of the words and phrases that the speech recognition engine will recognise. In order to setup speech recognition for the gaze and voice game, notifications for recognition events were first created. Then a `ISpRecoGrammar` was created, loaded and activated. This indicated which utterances to recognise so that when recognised the formerly mentioned events would be fired.

3.2.3 Game Development Tools

In order to see which tools would be most appropriate for game implementation an investigation was carried out. It was important to establish which tool would facilitate rapid game development while allowing for integration with both the Tobii SDK and SAPI.

Development Tool Investigation

Two game engines were looked at: The Torque Game Engine Advanced (TGEA) [44] and Unity [45]. TGEA was used by Kenny *et al.* [31] when creating their FPS game, although an EyeLink II eye tracker was used in that study, as described in Section 2.5.2. Rather than a trial version of the engine, only a demo was provided. The demo was very limited making it impossible to establish how easy it would be to integrate a game created using TGEA with SAPI or the Tobii SDK.

Unity did provide a trial version of their software. Although impressive, it was felt that the game engine was a highly abstracted tool making it difficult to see how easy

it might be to integrate with the Tobii SDK or SAPI.

The Object-Oriented Graphics Rendering Engine (OGRE) was also closely examined [46]. It is a scene-oriented, flexible 3D engine, as opposed to a game engine. OGRE is written in C++ and is designed to make it easier and more intuitive to produce applications utilising hardware-accelerated 3D graphics. It is an excellent open source graphics engine with a very active community and was shown to work with both voice recognition and a Tobii eye tracker by Wilcox *et al.* [38], as described in Section 2.6. The wealth of resources, active community and proven record of having worked with voice and gaze data in the past made it a good candidate with which to develop the application in.

XNA is a set of tools with a managed runtime environment provided by Microsoft that facilitates computer game development. The XNA framework is based on the native implementation of the .NET Compact framework 2.0. XNA is very user-friendly allowing for rapid development of games. To the author's knowledge, XNA has not been previously used in an eye tracking study. A test application was built to investigate if integration with gaze and voice was possible. The application successfully referenced both TetComp and SAPI.

It was decided to go for XNA as the development environment given that it integrated well with both TetComp and SAPI and using it would eliminate the extra time that would be required to learn OGRE.

3.2.4 The Component Object Model

Since the functionality of both SAPI and Tobii SDK will be exploited via COM it is timely to give a brief overview of COM and how it can be accessed from .NET.

COM Overview

COM is a standard specifying an object model and programming requirements that enable software components to interact with other objects [47]. A software object is usually made up of a set of data and functions that manipulate that data. A COM object is one in which access to an object's data is achieved exclusively through one or more sets of related functions. These functions are known as interfaces. The only way to gain access to the methods of an interface is through a pointer to the interface.

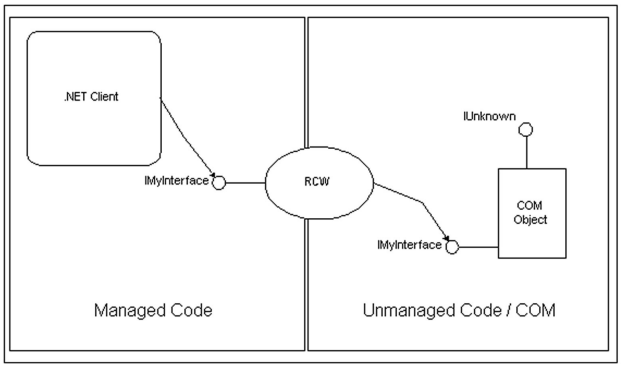

Figure 3.4: .NET runtime callable wrapper. Reprinted from [48].

A characteristic of COM is that it is Apartment Threaded. All COM objects in a given process are divided into groups known as *apartments* [47]. A COM object resides in only one apartment and its methods may only be directly called by a thread that belongs to that apartment. Other threads must go through a proxy. There are two types of apartments:

- **Single-threaded apartments:** Consists of exactly one thread. So all COM objects on that thread can only receive direct method calls from the thread which belongs to that apartment. All method calls are synchronized with the windows message queue for the single-threaded apartment's thread.
- **Multi-threaded apartments:** Consist of one or more threads. COM objects that reside in an multi-threaded apartment can only receive direct method calls from threads that belong to that apartment.

.NET/COM Interoperability

The procedure for calling COM components from .NET is shown in Figure 3.4. When a COM object is called from .NET, the runtime generates a runtime callable wrapper (RCW), which acts as a proxy for the unmanaged object [48]. The RCW binds to the underlying COM object, factoring the COM interface into a managed form. It is responsible for translating and marshalling data between the two environments.

3.3 Game Concept

The running of a user trial, described in detail in Section 5.1, has some implications on game design. First for the application to be useful some sort of mechanism for recording game data needs to be implemented. Second the duration of the trial must not be unreasonable since participants can become fatigued and eye tracking calibration can be lost [49]. Participants would need to fill in questionnaires and some time to familiarise themselves with the game controls. So with this in mind the game needed to be relatively simple so users could get get to grips with it quickly and finish it within a reasonable time frame. This would allow for a quick turn-around of participants. It was decided that the game should include as many common gaming tasks as possible, such as, navigation, object selection and so on. It was also desirable to make the game as inoffensive to potential participants as possible. To this end it was intended that the game have a fun and cartoon-like feel.

The premise decided upon was "Rabbit Run". The player is trapped in a rabbit warren, inhabited by evil bunnies, from which they must escape. The objective being to navigate through the maze and find the exit in the shortest time possible. In order to earn extra points coins distributed throughout the maze could be collected. The evil bunnies could also be shot. Once the exit was reached, players would gain their freedom and win the game. A map would also be provided in order to assist players finding their way through the maze. It was decided that the game would to be developed in the first-person perspective since this is how we view the world in our everyday lives.

3.4 Game Framework

To facilitate the development of a robust, easily updated application it was decided that a framework would be implemented as the basis for the application. This section describes the design of the main components of that framework. The `Game` class is the basis on which game projects are built in XNA and the use of `GameComponent`s and `GameService`s allows for the efficient organisation of any such project [50]. Where possible these features of XNA are been incorporated into the design of the game framework.

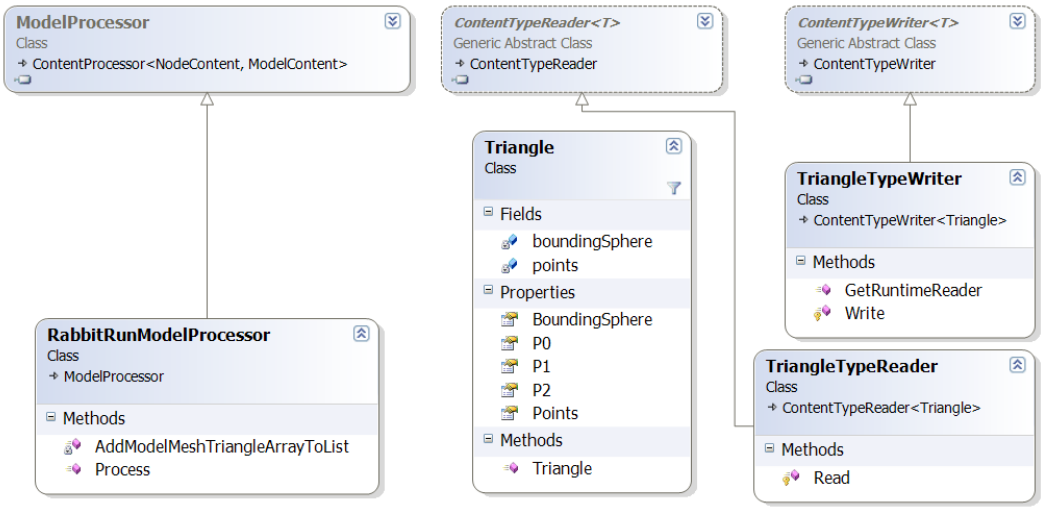

Figure 3.5: Class diagram for custom model processor.

3.4.1 Asset Loading System

In order to optimise the framework it was decided that a custom model processor would be used. Figure 3.5 shows the classes need to implement the processor. `RabbitRunModelProcessor` extends the XNA class, `ModelProcessor`, adding extra functionality in order to loop each vertex in each mesh of the model creating a list of `Triangles` which make up that mesh. Each `Triangle` is further processed to include the smallest `BoundingSphere` which encompasses its three vertices. Once created the `Triangle` list is added to `Tag` attribute of the mesh.

Making these calculations when loading the model optimises the code since the calculations need only be made once and stored within the loaded model. Storing a list of `Triangles` allows for collision detection tests.

3.4.2 Asset Management System

Figure 3.6 shows the main asset classes used in the system. There are four models to be used in the game. One to represent coins, one rabbits, one the maze and one the exit. Each model is represented by a class of its own, inheriting from the abstract class, `RabbitRunModel`. Most methods are concerned with updating and drawing the models. `Bunny` and `Coin` also define methods to allow for the shooting of rabbits and

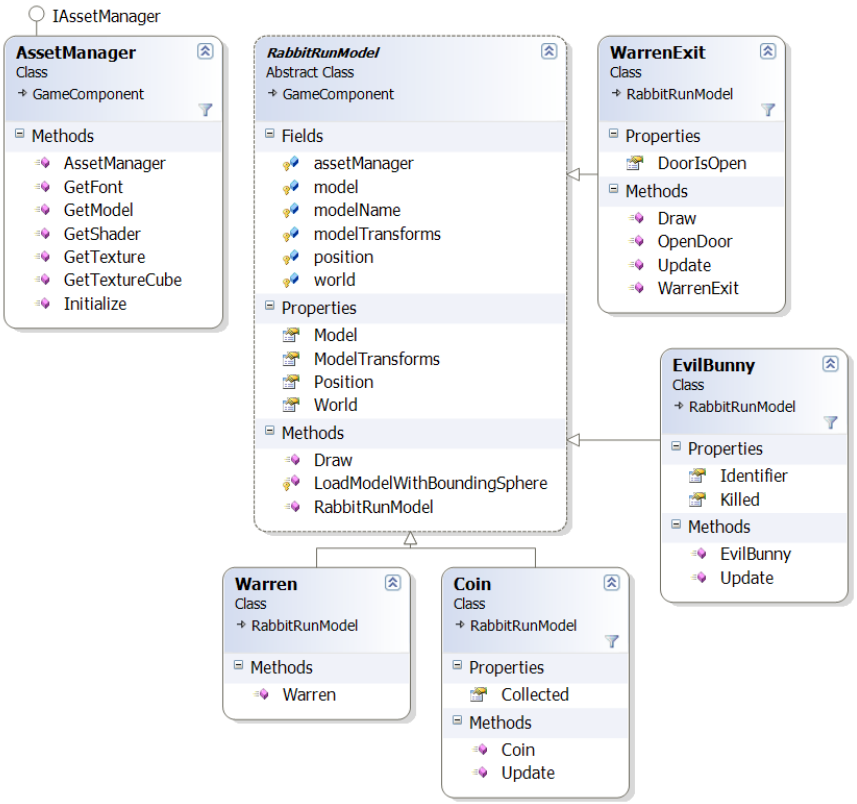

Figure 3.6: Class diagram of asset classes.

collection of coins (both of which gain extra scores).

`AssetManager` is intended to load and make available assets such as fonts, models, textures and shaders. It inherits from the `GameComponent` class and `IAssetManager` interface and is added as a `GameService`, which makes it readily available throughout the system.

3.4.3 Menu System

Figure 3.7 shows the menu classes of the system. There are menu classes for the various menu screens that will be required by the system, main menu, play menu, demo menu and pause menu. Each of these classes inherits from the abstract class, `Menu`. Each menu contains buttons which display the menu option, these are represented by the `MenuItem` class.

All menus are managed by the `MenuManager` class again this class is to be added as

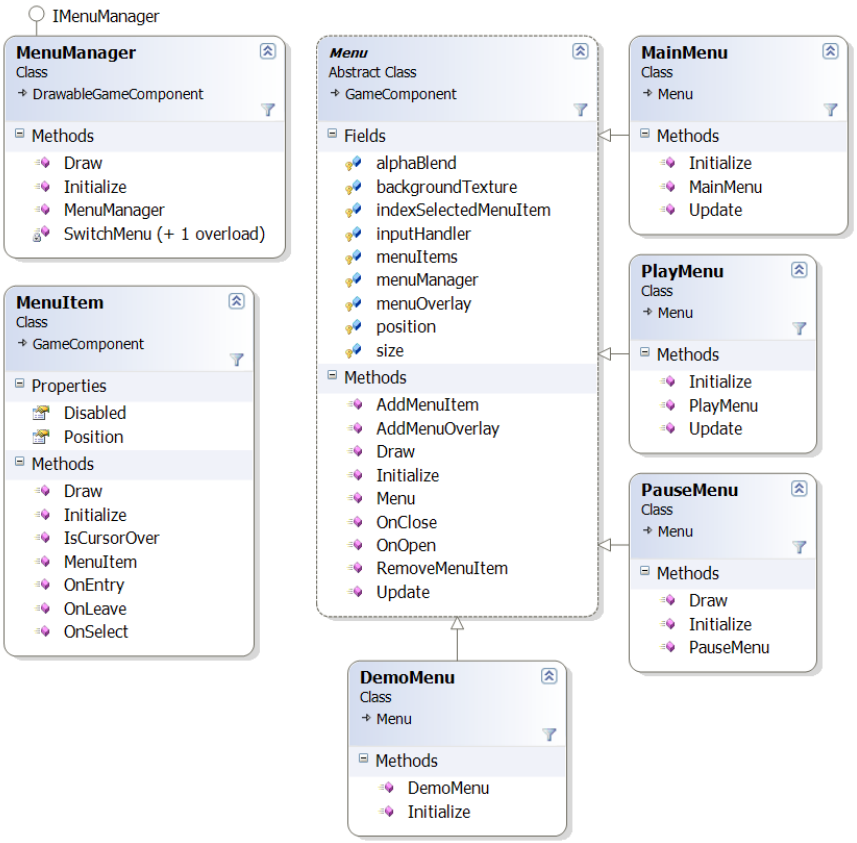

Figure 3.7: Class diagram of menu classes.

a `GameService`, which will make it readily available throughout the system.

3.4.4 Game Data Storage System

Figure 3.8 shows the classes and structs required to save relevant game data in XML format. The `GameDataManager` class provides methods which record various game data, such as shots fired, coins collected and so on. All of this information is stored in memory in the serialisable struct, `GameData`. When the game completes the `GameDataStorage` class is to be used to store that data in XML format.

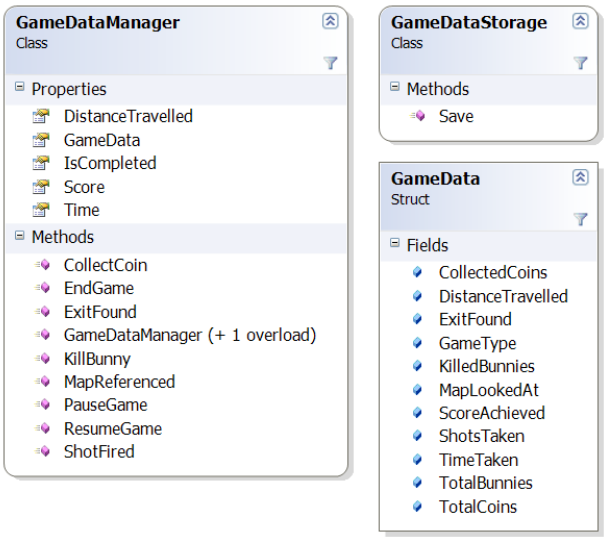

Figure 3.8: Class diagram of game storage classes and structs.

3.4.5 Map Generation System

Figure 3.9 shows the two classes required by the system in order to generate a map of the current maze. Once a game is loaded a new instance of `MapGenerator` is created. In doing so the the new instance of `MapGenerator` will parse the maze model building up a map of the model for use in the game. The map is a two dimensional array of `MapElement` objects. Initially the property `IsRevealed` is set to `false`. As the player travels around the maze this property will be set to `true`. Only `MapElements` with an `IsRevealed` property set to true will be rendered when displaying the map on screen.

3.4.6 Eye Tracking System

The `GazeInput` class, shown in Figure 3.10, creates a `TetClient` object and verifies if a compatible Tobii eye tracker is connected. If so, upon start up of the application, any calibration data present is cleared and a new calibration is initiated. Once calibrated, eye tracking is started. Events are fired by the `TetClient` when new gaze data is available. This data must then be relayed to the application. The gaze data includes the area being looked at on screen by both the left and right eyes and the distance of both eyes from the screen. This information is averaged to give the most likely *point-of-regard* (POR).

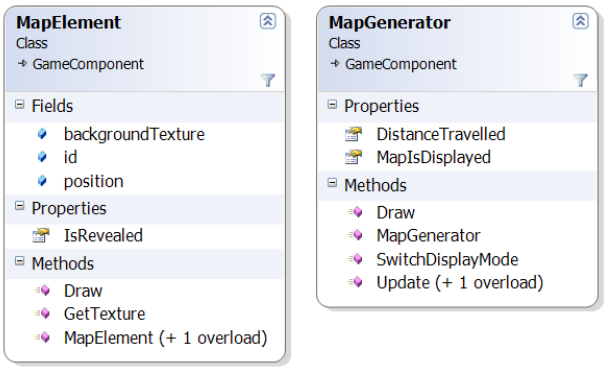

Figure 3.9: Class diagram of map generation classes.

3.4.7 Voice Recognition System

The `VoiceInput` class, shown in Figure 3.10, creates a `SpSharedRecoContext` object and attaches a grammar with specific grammar rules to it. These rules can be used to recognise certain words and phrases. These words represent the various voice commands which will be used in the system, such as "Option", "Walk", "Run" and so on. The recognition event, `_ISpeechRecoContextEvents_RecognitionEventHandler`, is fired when a voice command is recognised by the engine. This is used to inform the game that a player has issued a specific voice command.

3.4.8 Input Handling System

Figure 3.10 shows the four different input handler classes and the `InputAction` class. There are four input handler classes, one for each input type. When the application is first loaded the appropriate handler will be instantiated and added as a `GameService`, making it available throughout the framework. Each handler extends from the XNA `GameComponent` class. This means that the handler's `Update` method is called every update cycle of the application.

The `Update` method checks for user input and creates a list of `InputActions`. Other components in the game may access this list via the `PollInput` method. For example `MenuManager` might load a new menu depending on the `InputAction` list returned to it when calling the `PollInput` method.

Note that `GazeVoiceHandler` and `GazeKeyboardHandler` are to use `GazeInput` to

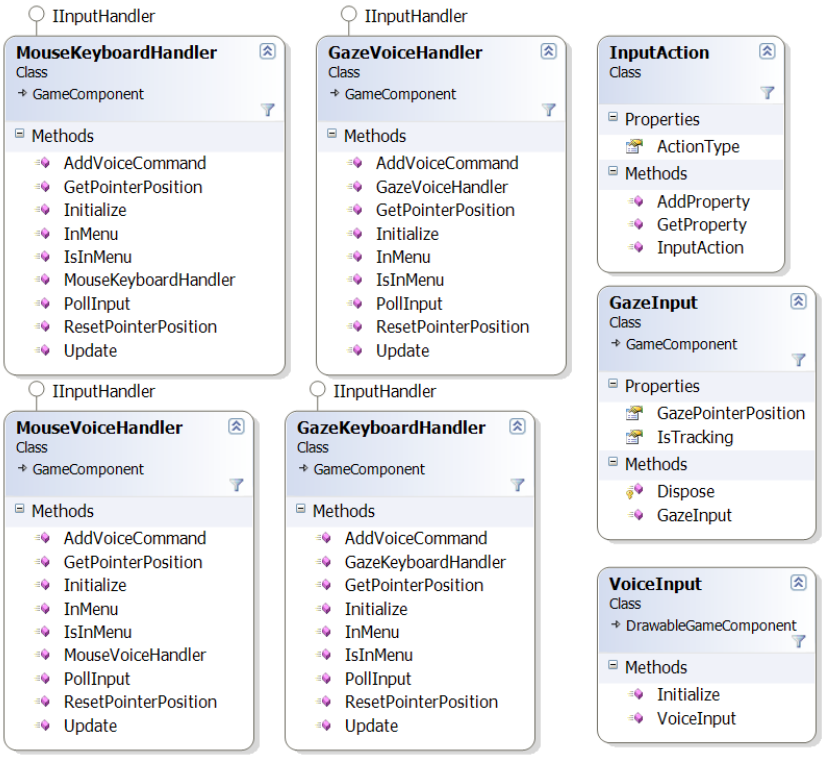

Figure 3.10: Class diagram of input classes.

access gaze data from the eye tracker. `VoiceInput` updates `GazeVoiceHandler` and `MouseVoiceHandler` directly when a speech recognition event is fired, see Sections 3.4.6 and 3.4.7 for more details.

3.4.9 Camera System

Figure 3.11 shows the camera classes of the system. Note that there are different cameras for each mode of input. This is because the different input types will control the cameras slightly differently from one another, see Section 4.2 for more details. Since the game is to be in the first person perspective, the camera in effect represents the player. Therefore the motion of the camera must be checked for collisions. The `GetBoundingSphere` and `GetBigBoundingSphere` methods provide positional information for collision detection between the camera and the surround game models.

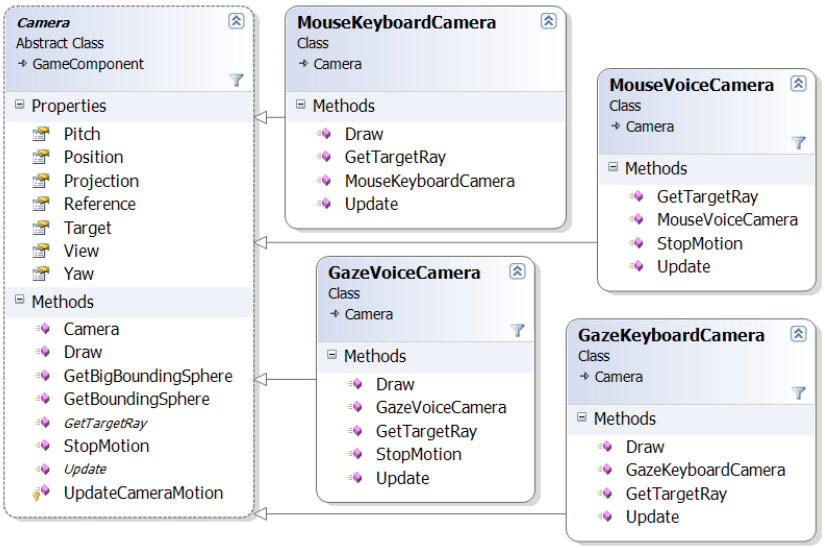

Figure 3.11: Class diagram of camera classes.

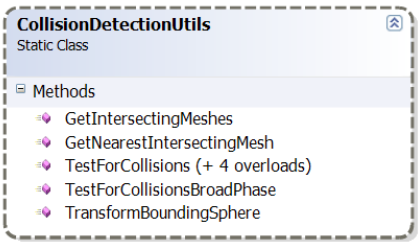

Figure 3.12: Static collision detection utility class.

3.4.10 Collision Detection System

The static class `CollisionDetectionUtils` is the main collision detection for the application and is shown in Figure 3.12. The method `TestForCollisionsBroadPhase` and some overloaded methods of `TestForCollisions` allow for testing of collisions between the camera and the various models, the warren, the exit, rabbits and coins. The collision response is dealt with by the calling object, the purpose of `CollisionDetectionUtils` is to merely test if a collision occurred or not.

`CollisionDetectionUtils` also provides an overloaded method of `TestForCollisions` which tests for collisions between a `Ray` and a model. This is intended to check which model a bullet collides with, and therefore hits. Thus helping to determine whether or not a rabbit has been shot.

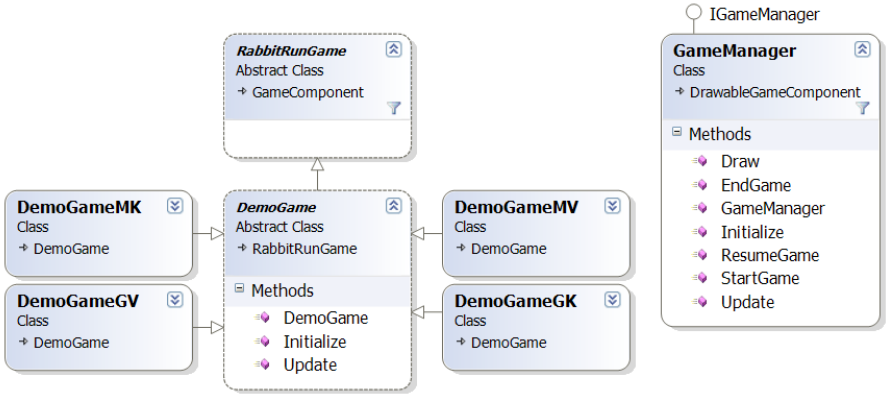

Figure 3.13: GameManager and demo classes.

3.4.11 Game Management System

The demo game classes are shown in Figure 3.13 to illustrate how the game class hierarchy is structured and how the `GameManger` class interacts with game classes. The abstract class `RabbitRunGame` is at the top of the hierarchy all sub-classes inherit its functionality. This is a three tier hierarchy, on the next level down classes define where the initial camera positions are, where coins, rabbits and the exit are to be placed through out the maze. In this case `DemoGame` resides on this tier. On the next level of the hierarchy are the input method specific game classes, in this case `DemoGameMK`, `DemoGameGV`, `DemoGameGK` and `DemoGameMV`. The suffixes `MK`, `GV`, `GK` and `MV` denoting "Mouse/Keyboard", "Gaze/Voice", "Gaze/Keyboard" and "Mouse/Voice" respectively. There is a need for separate classes for each type of input as each game will use a different camera (as already discussed in Section 3.4.9) and be controlled and react differently.

Figure 3.13 also shows the `GameManger` class. All games are managed by the `GameManger` class and again this class is added as a `GameService`, making it available throughout the system. The `StartGame` method creates a new instance of the selected game class and adds it as a `GameComponent`, this method is to be invoked from the menu. `GameManger` also provides methods to end, pause and resume games.

Chapter 4

Implementation

This chapter discusses the implementation of the gaze and voice game. Section 4.1 gives an overview of the main features of the game framework and how it operates. Then Section 4.2 details how cameras where implemented and how they work within the framework. Finally Section 4.3 examines issues which arose during the implementation.

4.1 Framework Implementation

`RabbitRunMain` is the main class of the application. Figure 4.1 shows this class and the other main classes used by the application. When `RabbitRunMain` is constructed it carries out the following steps:

- Create an input handler (dependent on type of input required).
- Create a new `AssetManager`.
- Create a new `GameManager`.
- Create a new `MenuManager`.

Each of which are added as `GameComponent`s automatically including them in the update cycle of the application and, should they be `DrawableGameComponent`s, also in the draw cycle.

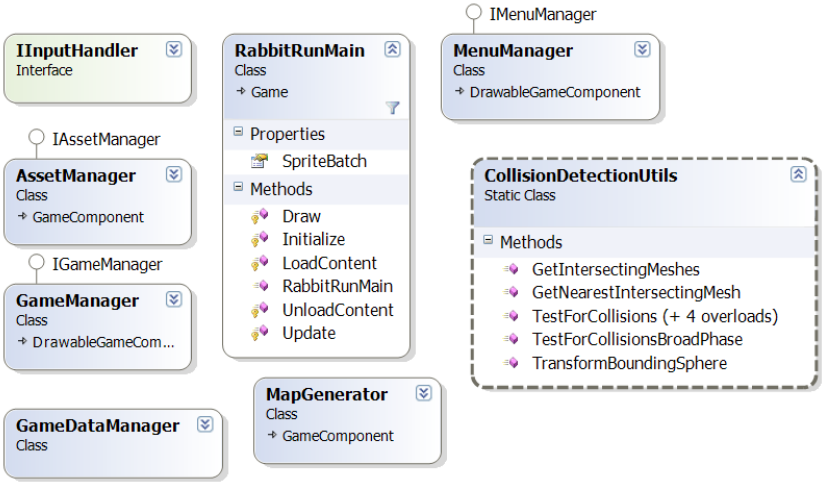

Figure 4.1: Class diagram showing main classes.

4.1.1 Update Cycle

While menus are being displayed only `InputActions` relating to the menu are processed by the input handler. These actions are to be returned to the `MenuManager` via the `PollInput` method. These actions allow for menu navigation.

Once a user selects a game to play, from the menu, only `InputActions` relating to the game are processed by the input handler. These actions are returned to the loaded game, again using the `PollInput` method. During the update cycle the currently loaded game must do following:

- Update each displayed `Bunny`, running a simple hopping animation.
- Update each displayed `Coin`, rotating the coin.
- Update the camera, processing user input and updating the camera position and orientation accordingly.
- Check for collisions between the camera and `Warren` and `Bunny` model classes.
  - React to any such collisions.
- Check for collisions between camera and `Coin` models.
  - Updating score and removing coin should collision occur.
- Check for collisions between camera and `WarrenExit` model.
  - Triggering end game should collision occur.

- Check for fire bullet `InputAction` and animate bullet firing.
- Check for bullet intersection with `Bunny` model class.
  - Updating score and removing rabbit should collision occur.
- Update the `MapGenertor` with the player's current location.

Also during the update cycle the `GameDataManager` will update its record of relevant game data.

4.1.2 Draw Cycle

While the menu is being displayed the `Draw` method of the `MenuManager` class will be invoked. `MenuManager` selects the currently displayed menu and invokes it's `Draw` method, thus displaying that menu.

Similarly while a game is being played the `GameManager` class calls the currently selected game's `Draw` method. The game's `Draw` method will render the various models in their current positions and orientations.

4.1.3 Threading

In order to correctly synchronise XNA with TetComp, the keyword `[STAThread]` was used at the main entry point to the application. This tells XNA to use single-threaded apartment (STA) threading when dealing with COM objects created in the application.

Because multi-threaded apartment (MTA) is more efficient than STA threading it is selected by default by .NET when communicating with COM objects. However since STA in COM is single-threaded no additional thread synchronisation is required. Calls to objects on STA threads are serialised with Windows message handling on that thread. This makes sure that both the COM objects and the underlying windowing objects are all synchronised.

Without STA enabled TetComp bombarded the XNA application with events from the eye tracker bringing the frame-rate down to 1 frame-per-second, making the application unusable.

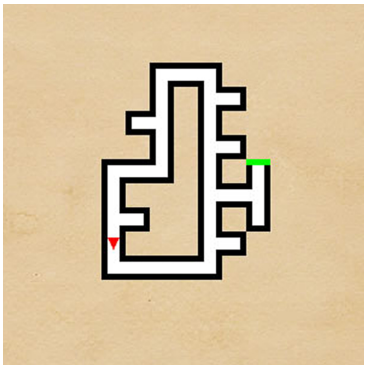

Figure 4.2: Example of a map displayed in the game. In this example the entire map has been revealed.

4.1.4 Map Generation

The `MapGenerator` class was implemented as set out in Section 3.4.5. However there was a subtle difference between its implementation using mouse input versus gaze input. For mouse input, the map was only updated with the coordinates of where the player currently was. So if the player traveled to a new location that location would be revealed on the map. Figure 4.2 shows a map from the demo game, the red arrow indicates the avatar's current location and direction, the green bar indicates where the exit is located.

For gaze input a novel game feature was developed. It was decided to reveal the locations where the player looked. So if the player looked at a particular location that area would be revealed on the map without requiring the player to actually move to those positions. This feature could be useful in puzzle based games where the player needs to memorise parts of the virtual environment.

4.2 Camera Implementation

Given that the game is implemented in the first person perspective, the cameras in the game plays a dual role of showing the game play and acting as the player's avatar. As such it is appropriate to discuss in some detail the implementation of the cameras in the application and how they relate to the different input types.

4.2.1 Mouse Camera

The cameras `MouseKeyboardCamera` and `MouseVoiceCamera` are almost identical and so are discussed as one in this section. In each update cycle the mouse was repositioned at the centre of the game screen. To update the camera view the user could move the mouse in the direction they wished the camera to face. The distance between the centre point and the mouse pointer screen coordinates where used to calculate the camera pitch (difference in Y value) and the yaw (difference in X value).

The different input from either keyboard or voice commands determined if the camera position itself was to change. If such input was received the camera position was updated accordingly. The camera target was then updated with appropriate rotations from both pitch and yaw and transformed using the new camera position. Finally the view matrix was updated using the new position and target.

Mouse Camera Targeting

For targeting rabbits to shoot a cross hairs was rendered at the current position of the mouse, which is almost always at the centre of the screen. So in order to shoot a rabbit players must move the camera (by shifting the mouse) to centre the target at the centre point of the screen. This is the case with most FPS games available today. The screen coordinates can then be unprojected to define a ray along which the centre point travels through into the 3D virtual game world. This information can be used to fire bullets along that ray.

4.2.2 Gaze Camera

Again the cameras `GazeKeyboardCamera` and `GazeVoiceCamera` are similar enough to be discussed together in this section. The idea adapted for creation of a gaze input camera builds upon the idea of Castellina and Corno [37] of using semi-transparent buttons to rotate the camera and move the avatar, see Section 2.5.2 for more details. As shown by Kenny *et al.* [31] the vast game time in a FPS games is spent looking in the inner 400 x 300 rectangle of a 800 x 600 resolution screen, see Section 2.5.2 for more details.

The idea was to utilise the outer rectangle of the screen to place semitransparent

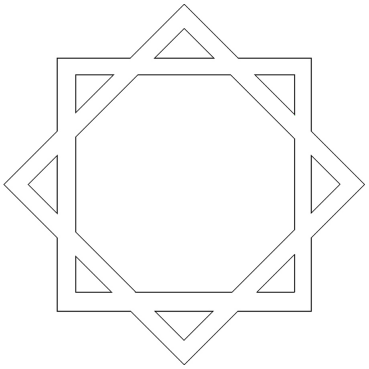

Figure 4.3: Eight-sided Armenian Star.

gaze activated buttons. The inner rectangle could be left alone to allow normal game interaction. By placing the buttons in this outer area it was hoped that they would not interfere with game play. Also the buttons would not be displayed unless activated, by looking in that area of the screen, to avoid distracting the player.

The purpose of the buttons was to rotate the camera in a given direction. So by looking at the upper part of the screen the camera would shift upwards, if looking at the bottom of the screen the camera would shift down, if looking left the camera would shift left and so on. The original idea was to place the buttons in such a way as to form an eight-sided Armenian star, as shown in Figure 4.3. The buttons would act as a visual aid to the player indicating the direction the camera was shifting. Figure 4.4 shows the eight different camera shifting buttons as they were displayed in the game.

Gaze Camera Targeting

A cross hairs was rendered where the current gaze screen coordinates were on screen. This separated the targeting from the camera view much in the same way as Jönsson [32] did in her Half Life demo. The screen coordinates of the cross hairs are unprojected in the same way as for mouse targeting. This information is again used to calculate the trajectory of bullets.

4.2.3 Updating Camera Position

Two different input types, voice and keyboard, were used to update the camera position in the game. This section describes how those input types were used to implementation

Figure 4.4: Gaze camera buttons as displayed in game.

camera motion and hence simulate player navigation.

Keyboard Input

Navigation in the game environment was implemented using the arrow keys since this would be the most intuitive even to a novice player. So players could move forwards, backwards, left or right relative to the direction the camera was pointing. This updated the position of the camera at a "walking" pace. If players wanted to increase their speed they needed to hold down the shift key.

Voice Input

Navigation with voice recognition input was achieved using three commands "Walk", "Run" and "Stop". When the "Walk" command was issued the camera proceeded to move, at a walking pace, in the direction the camera was facing until such time as it encountered an obstacle such as a wall or a rabbit or the "Stop" command was issued. The "Run" command was implemented to operate in a similar way except at a running pace.

4.3 Implementation Issues

Some problems were encountered when implementing voice recognition. More intuitive voice commands were not always recognised so more distinct voice commands were chosen. For example instead of saying "Map" to bring up the game map, "Maze" had to be used instead. When selecting menu items "Select" too proved to be inconsistent so the command "Option" was used instead.

A pilot user trial involving one participant uncovered a number of issues which were not anticipated prior to that time. It was decided to to delay the user trials until these issues were resolved to some degree. The participant in the pilot study found the the eight-pointed star gaze camera buttons unusable. She was unable to control the camera well and the camera rotated when she did not want it to, causing it to spin in a disorientating manner. This was most likely to the narrow screen of the integrated eye tracker monitor used in the project. Perhaps if a wider screen was used this issue may not have occurred. Due to time constraints it was decided to only use the left and right buttons as shown in Figure 4.5 rather than to recode them.

Originally it had been intended to include obstacles over which players would have to jump over or crouch under. However the pilot study showed these were difficult to navigate through so it was decided to remove these obstacles from the game. There was not enough time to update the code to make them easier to navigate through.

The collision response also proved to be poor with the player getting "caught" near walls. Due to time constraints it was decided that there would not be enough time to recode this so a quick fix was settled upon. The fix was to stop any voice commands such as "Walk" or "Run" so that the player would stop colliding with the wall they

Figure 4.5: Left and right gaze camera buttons.

were facing.

The game also proved to be more time consuming than initially anticipated so a time limit of eight minutes was placed in the game so that trial would not take too much of the participants time.

Chapter 5

Evaluation

This chapter examines the created gaze and voice game and gives an evaluation of how successful it was. Section 5.1 describes how the user trial was designed and run. Then Section 5.2 goes on to review the results gathered during the user trial. Finally Section 5.3 presents an appraisal of those results.

5.1 Experimental Design of User Trial

One of the main objectives of the project was to gather useful information from which conclusions could be drawn, such as:

- How does gaze and voice compare with keyboard and mouse as a means of interaction?
- How enjoyable is the use of gaze and voice as a means of interaction?

The first objective is relatively easy to quantify. Data could be gathered and saved while participants played the game see Section 3.4.4 for details on data storage. The second is more subjective and requires the use of questionnaires to gather participants' opinions.

In order for the data to be useful comparisons must be made between different types of input. This means participants would need to play the game using both modes of interaction. They therefore played the game using mouse and keyboard as well as with gaze and voice.

Another consideration was, if gaze and voice proved not to be a viable form of interaction, which mode of interaction was at fault. So ideally data needed to be gathered to resolve this. The best way to answer this possible issue would be to make a second comparison of gaze and keyboard against mouse and voice. This meant asking participants to play the game four different times, with the four different combinations of interaction.

- Gaze & Voice
- Mouse & Keyboard
- Mouse & Voice
- Gaze & Keyboard

This approach has two major drawbacks. The first drawback concerns the learning effect. Players are likely to improve at the game the more they play it. So by the time participants play using the fourth type of interaction, it would be difficult to determine if the means of interaction has benefited the player or if it was simply due to their skill level improving over time.

The second drawback was time, it would be unreasonable to ask a participant to spend longer than twenty minutes completing a trial. Anything longer and they could begin to lose interest skewing results. Also the longer the duration of the trial the more likely eye tracker calibration would be lost. However given the need for two different comparisons, involving four different types of interaction it was decided that there should be two groups of experiments as follows:

- Mouse & Keyboard versus Gaze & Voice
- Gaze & Keyboard versus Mouse & Voice

5.1.1 Game Layout

Given that the game would need to be played twice and that results would need to be comparable, the game layout in each version would also need to be comparable. To this end it was decided that the exact same layout would be used in each trial by only swapping the start and exit points. The exact same number of coins and rabbits were used, distributed in the same positions in each setup. Using the same basic layout for each trial would yield comparable results. While having different start and end points

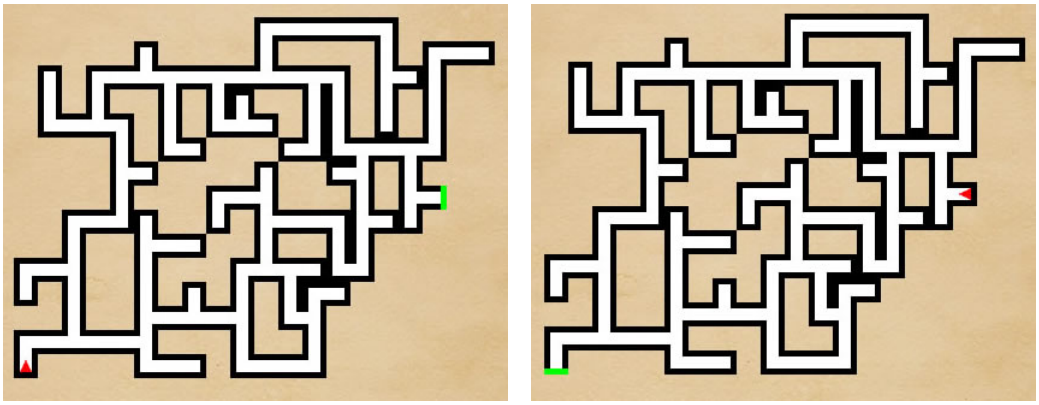

Figure 5.1: Different maze setups for user trials.

Participant	**Experiment #1**	**Layout**	**Experiment #2**	**Layout**
1	Mouse & Keyboard	Setup 1	Gaze & Voice	Setup 2
2	Gaze & Voice	Setup 1	Mouse & Keyboard	Setup 2
3	Gaze & Voice	Setup 2	Mouse & Keyboard	Setup 1
4	Mouse & Keyboard	Setup 2	Gaze & Voice	Setup 1
5	Mouse & Keyboard	Setup 1	Gaze & Voice	Setup 2
6	Gaze & Voice	Setup 1	Mouse & Keyboard	Setup 2
7	Gaze & Voice	Setup 2	Mouse & Keyboard	Setup 1
8	Mouse & Keyboard	Setup 2	Gaze & Voice	Setup 1
9	Mouse & Keyboard	Setup 1	Gaze & Voice	Setup 2
10	Gaze & Voice	Setup 1	Mouse & Keyboard	Setup 2

Table 5.1: Participant ordering for mouse & keyboard versus gaze & voice user trial.

would eliminated the possibility of skewed results from learning. Figure 5.1 shows the alternate layouts which were used in each user trial.

Counter balancing was used so both the layout and the input types were alternated. Table 5.1 shows the order in which user trials were conducted for the Mouse & Keyboard versus Gaze & Voice comparison trial. It was hoped that alternating the ordering in this way would eliminate any distortion of results if a particular maze setup were easier than another.

Figure 5.2: Hardware setup for user trial.

5.1.2 Experimental Setup

The user trial took place in a sound proof video conferencing room and ran over four nonconsecutive days. This was to avoid any interference background noise might have on voice recognition. The hardware setup is shown in Figure 5.2. It consisted of a laptop running the application while connected to the Tobii T60 eye tracker (with integrated monitor) via an Ethernet connection. A keyboard, a mouse and a microphone headset were also connected to the host laptop to allow for keyboard, mouse and voice input. An adjustable chair was provided to allow participants to make themselves comfortable and place themselves 60 cm from the eye tracker monitor.

5.1.3 Questionnaires

When participants arrived they were asked to complete a consent form and answer a short background questionnaire about age, profession and their gaming habits, as shown in Appendix A.1. If any participant suffered from epilepsy they were to be excluded from the trial as a precaution. Immediately after playing each of the two games in the trial, participants were asked to answer another questionnaire, as shown in Appendix A.2. This questionnaire aimed to get the participants opinions on how easy or difficult they found that particular input. It was given immediately after playing the game so that any opinions they had would be fresh in their minds. After the second and final game was played (and the post-trial questionnaire was completed) a third and final questionnaire was given to the participants, as shown in Appendix A.3. This final questionnaire aimed to compared the two different games played by the participant to

gauge which they preferred. An open question at the end invited comments from the participants.

5.1.4 Other Materials

In order for all participants to get the same instructions it was decided that an instruction pamphlet would be used to inform all volunteers in a consistent way. A pamphlet was created for each type of input as shown in Appendices B.1- B.4.

5.1.5 Participants

Participants were sought by sending an email to postgraduates and staff in Trinity College. Fourteen people volunteered for the trial. Of these, thirteen users successfully completed the experiment two women and eleven men aged between 23 and 38. One user was excluded after running into difficulty maintaining calibration throughout the experiment. Of the thirteen who completed the trial two participants encountered notable difficulty using gaze, one wore glasses and the other contact lenses. It was decided to also exclude these participants from the study. All other participants had normal or corrected-to-normal vision. It was felt that because of the low numbers of participants it would be best to use as many as possible in the first study (measuring Mouse/Keyboard versus Gaze/Voice) so that statistically relevant data could be gathered. To this end eight participants took part in the first study while only three took part in the second (comparing Gaze/Keyboard versus Mouse/Voice). Ideally another five would have allowed for further comparisons.

5.1.6 Procedure

To create the same environment for each participant the following procedure was used for the user trial for each participant:

1. A consent form was given to and signed by the participant.
2. The background questionnaire was given to and answered by the participant.
3. The participant was given the relevant instructions pamphlet.

4. The participant was asked to play the demo version of the game for as long as they wanted and encouraged to ask any questions they might have.

5. Once happy with the controls participants were asked to complete the full user trial version of the game in the room by themselves.

6. The post-trial questionnaire was given to and answered by the participant.

7. Steps 3-6 were repeated for the second experiment.

8. The final comparison questionnaire was given to and answered by the participant.

5.2 Results

This Section examines the results gathered in the user trial both from game data and questionnaires. A complete set of the results gathered is tabulated in Appendix C.

5.2.1 Mouse/Keyboard Versus Gaze/Voice

The results presented in this Section have been subdivided into related categories.

Performance

It is perhaps difficult to quantify how well a player did in the game. There were three objectives, to find the exit in the fastest time possible, to collect coins and shoot bunnies. So to measure performance it is necessary to examine each of these factors. Figure 5.3 shows this data graphed.

Related t-tests were used to see if there was statistically significant differences between these performance measures. The difference between distances traveled using Mouse/Keyboard (mean = 122.5) versus using Gaze/Voice (mean = 73.25) was significant (T $(df = 7) = 2.556$, $P < 0.05$, two tailed). The difference in time taken to finish the game was also significant showing that participants finished quicker using Mouse/Keyboard (mean = 228472.125 ms) than when using Gaze/Voice (mean = 418864.4 ms), (T $(df =7) = 4.683$, $P < 0.01$, two tailed).

Since there was a time limit of eight minutes imposed it was decided to also measure the speed (distance covered divided by time taken) to see if players covered less distance

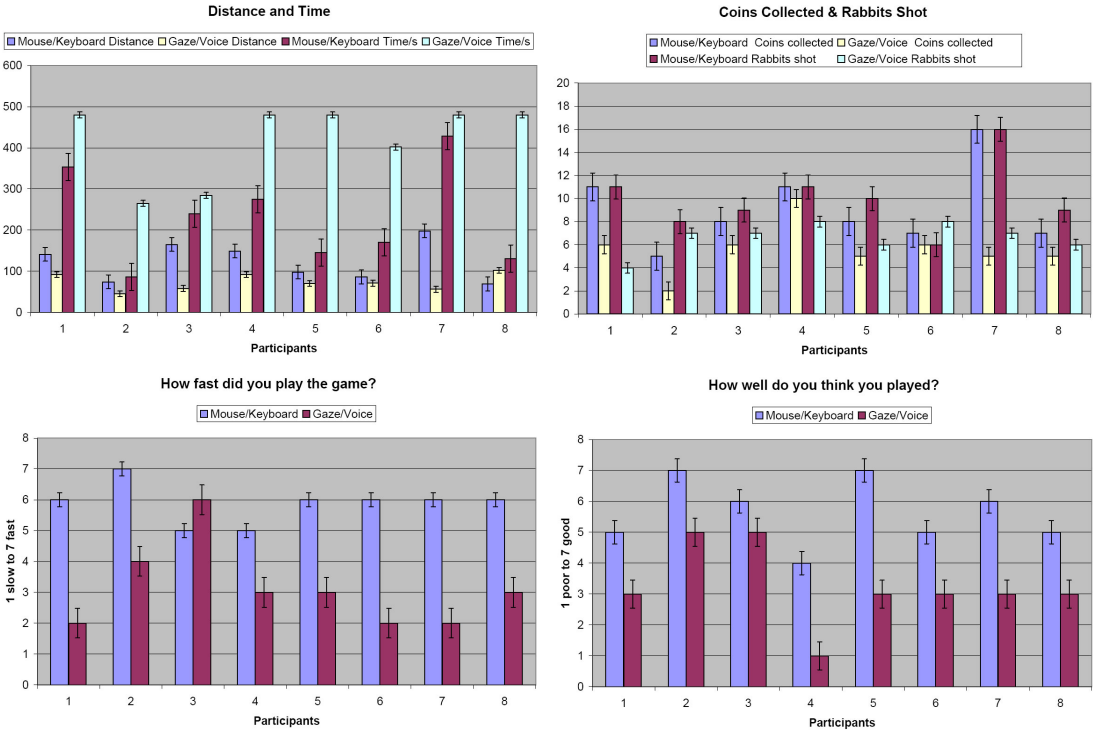

Figure 5.3: Graph of performance measures. Top left shows distance covered and time taken to complete the game and top right shows number of coins collected and rabbits shot. Bottom left shows how fast participants felt they played and bottom right how well they thought they played.

using Gaze/Voice. A related t-test showed speed to be higher using Mouse/Keyboard (mean = 0.583) versus using Gaze/Voice (mean = 0.176) (T (df = 7) = 7.4961, $P <$ 0.001, two tailed). The difference was statistically significant.

The difference between rabbits shot and coins collected was also shown to be statistically significant. Using Mouse/Keyboard a mean of 10 rabbits where shot as opposed to 6.625 using Gaze/Voice, (T (df = 7) = 2.791, $P < 0.05$, two tailed). The difference in the of number of coins collected using Mouse/Keyboard (mean = 9.125) versus using Gaze/Voice (mean = 5.625) was also statistically significant (T (df = 7) = 3.004, P < 0.02, two tailed).

In a two-tailed Wilcoxon Signed-Ranks test, participants felt they played faster using Mouse/Keyboard (median = 6) as opposed to with Gaze/Voice (median = 3), ($T = 1$, $N = 8$, $0.01 < P < 0.02$). They also felt they performed better overall using

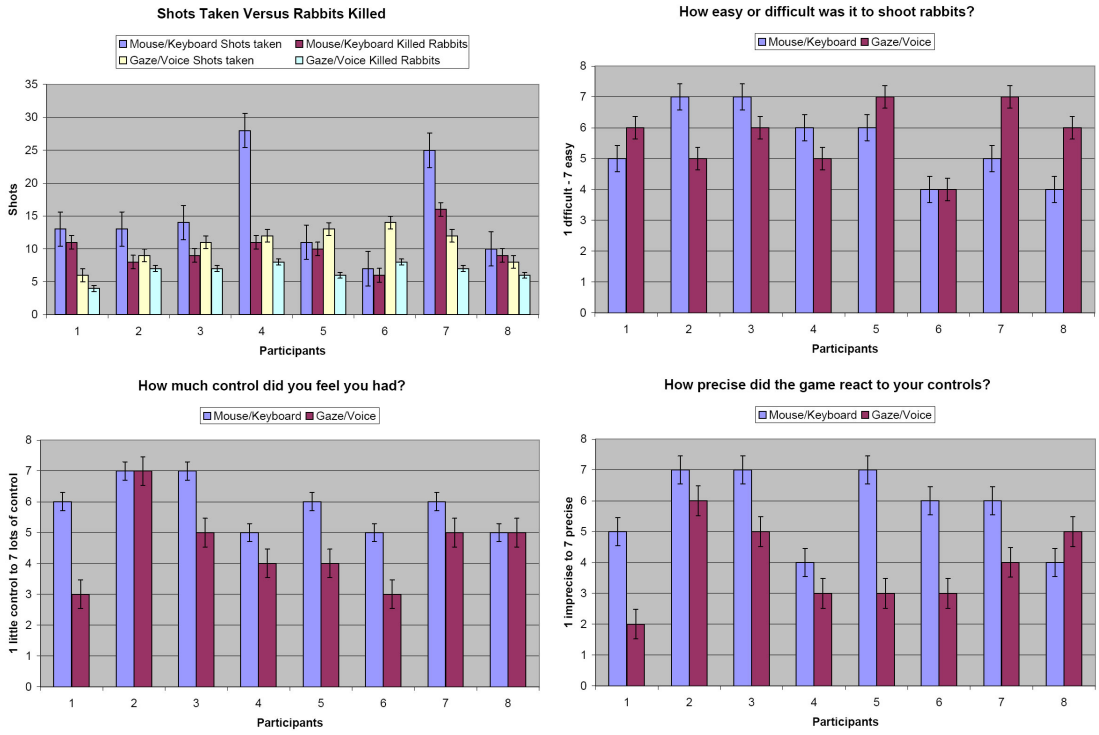

Figure 5.4: Graph of accuracy and control measures. Top left shows number of rabbits killed versus shots taken and top right shows how difficult participants felt it was to kill rabbits. Bottom left shows how much control users felt they had while bottom right shows how precise that control was.

Mouse/Keyboard (median = 5.5) versus Gaze/Voice (median = 3), ($T = 0$, $N = 8$, $0.01 < P < 0.01$). So participants performed worse using gaze and voice input across all measures.

Accuracy & Control

Figure 5.4 shows graphs of shots taken versus rabbits killed and how difficult participants felt it was to shoot rabbits as well as how much control was felt by participants and how precise it was. To gauge accuracy the number of shots taken was measured against the number of rabbits killed. In a related t-test no statistically significant difference was established between shooting accuracy using Mouse/Keyboard (mean = 72.6%) versus using Gaze/Voice (mean = 64%)(T ($df = 7$) = 1.034, $P > 0.2$, two tailed). Again this is backed up by a two-tailed Wilcoxon Signed-Ranks test which

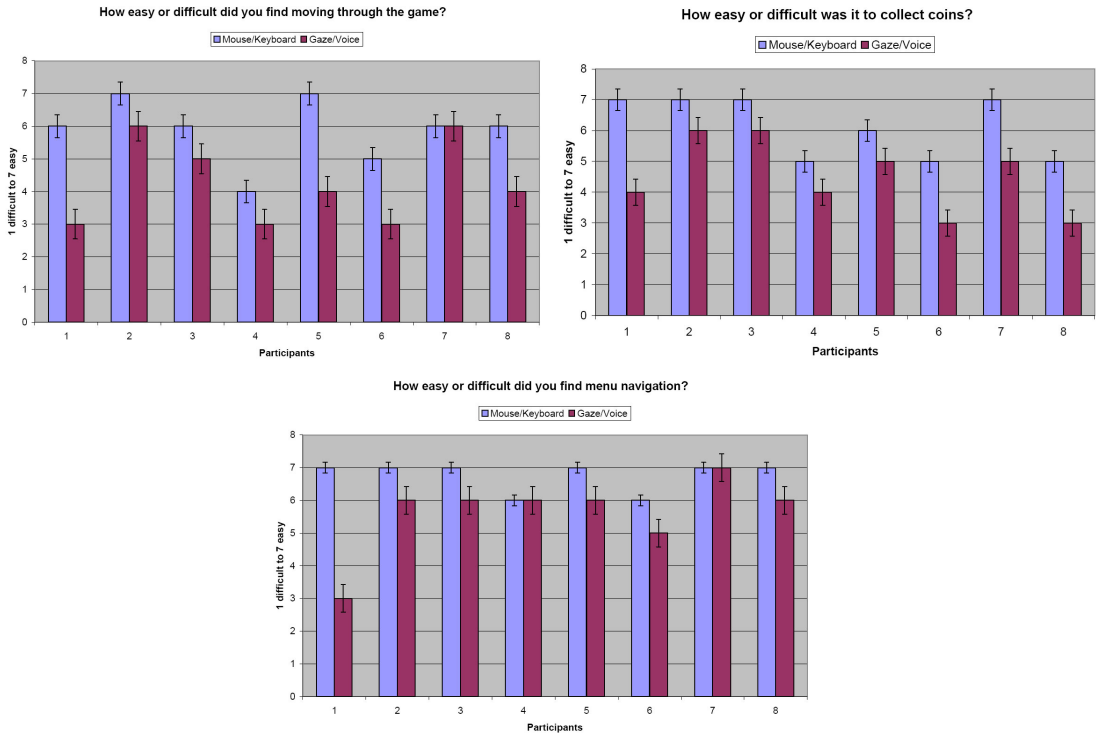

Figure 5.5: Graph of navigation measures. Top left shows how easy or difficult it was to navigate around the game, top right how difficult or easy it was to collect coins. Bottom shows how easy or difficult participants found menu navigation.

showed there was no statistically significant difference between how difficult it was precieved to shoot rabbits using Mouse/Keyboard (median = 6) or with Gaze/Voice (median = 6), ($T = 11$, $N = 7$, $P > 0.2$).

A two-tailed Wilcoxon Signed-Ranks test shows that participants felt Mouse/Keyboard (median = 6) to offer more control than with Gaze/Voice (median = 4), ($T = 0$, $N = 6$, $P < 0.001$). They also felt that the control was more precise for Mouse/Keyboard (median = 6) than with Gaze/Voice (median = 3.5), ($T = 2$, $N = 8$, $0.02 < P < 0.05$).

Navigation

A two-tailed Wilcoxon Signed-Ranks test showed game navigation was perceived to be significantly easier using Mouse/Keyboard (median = 6) than when using Gaze/Voice (median = 4), ($T = 0$, $N = 7$, $P < 0.001$). Coin collection can also be thought of as

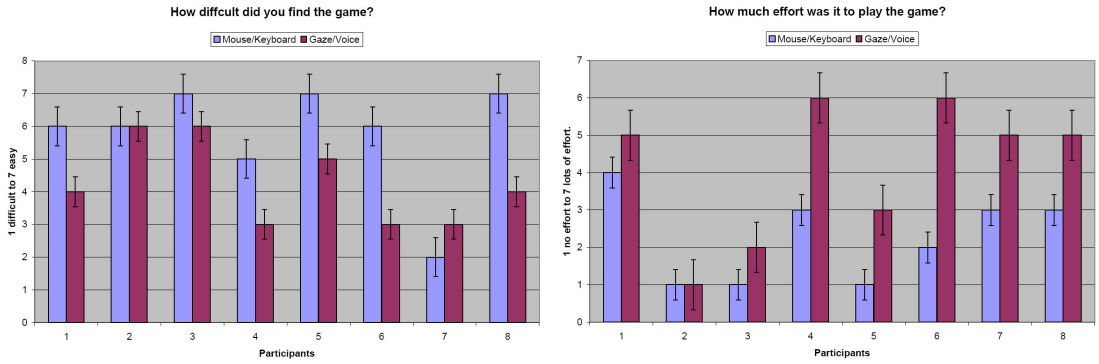

Figure 5.6: Graph of difficulty measures. Left shows ranking of difficulty, right the amount of effort required to play the game.

an indirect measure of how easy it was to navigate through the game. Coins were all placed in small corner areas of the maze were navigation would be most testing. In a two-tailed Wilcoxon Signed-Ranks test coin collection was found to be easier using Mouse/Keyboard (median = 6.5) than when using Gaze/Voice (median = 4.5), ($T = 0$, $N = 8$, $P < 0.001$). Figure 5.5 shows graphs of these navigation measures.

Again in a two-tailed Wilcoxon Signed-Ranks test menu navigation using Mouse/Keyboard (median = 7) was ranked easier than with Gaze/Voice (median = 6), ($T = 0$, $N = 6$, $P < 0.001$). Figure 5.5 also shows how menu navigation was ranked by the participants. The participant who ranked menu navigation as being most difficult commented on it saying "Calibration slightly off in menu but not so apparent in game". So perhaps the calibration for this participant could have been better.

Game Difficulty

Figure 5.6 shows how difficult participants found the game. In a two-tailed Wilcoxon Signed-Ranks test participants found the game to be less effort to play using Mouse/Keyboard (median = 3) as opposed to with with Gaze/Voice (median = 5), ($T = 0$, $N = 7$, $P < 0.001$). Mouse/Keyboard was also ranked easier (median = 6) as opposed to with Gaze/voice (median = 4), ($T = 1.5$, $N = 7$, $0.02 < P < 0.05$).

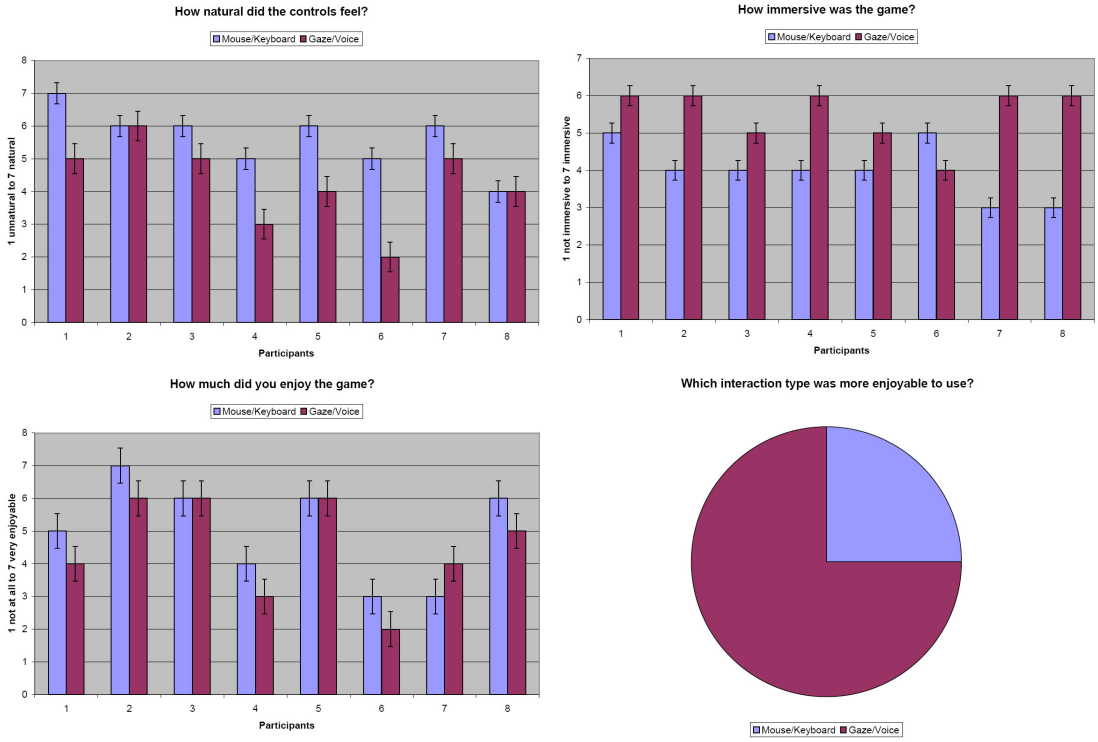

Figure 5.7: Graph of immersion, naturalness and enjoyment measures. Top left graphs how natural the control was ranked and top right shows how immersive the input type was ranked. Bottom left shows how enjoyable the game was rated after each trial, while bottom right shows which input type users felt was more enjoyable.

Game Enjoyment

While participants thought using Mouse/Keyboard (median = 6) was more natural than using Gaze/Voice (median = 4.5), ($T = 0$, $N = 6$, $P < 0.001$), Gaze/Voice (median = 6) was ranked as being more immersive than Mouse/Keyboard (median = 4.5), ($T = 2.5$, $N = 8$, $P < 0.05$). A Wilcoxon Signed-Ranks test was used in both cases. Figure 5.7 shows the graphs of both sets of data.

Figure 5.7 also shows graphs of enjoyment measures. A two-tailed Wilcoxon Signed-Ranks test showed no statistically significant difference between Mouse/Keyboard (median = 4.5) and Gaze/Voice (median = 4), ($T = 3.5$, $N = 6$, $P > 0.2$). However when asked which they preferred, 75 % of participants selected Gaze/Voice as the more enjoyable.

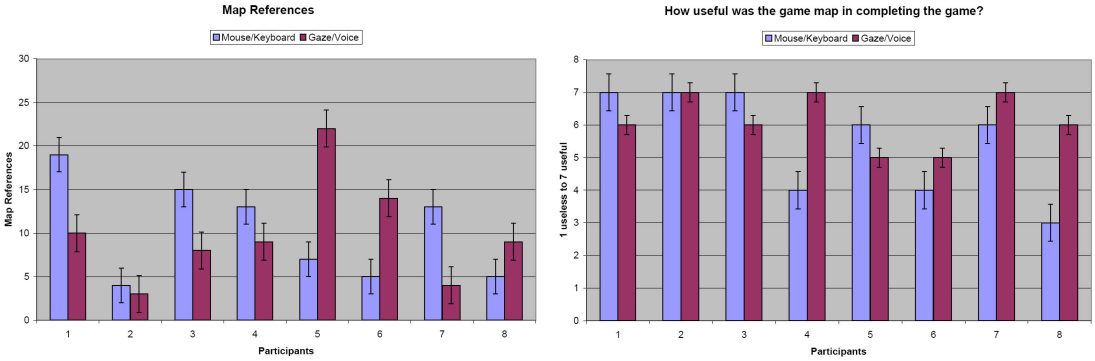

Figure 5.8: Graphs of map references and map usefulness in trial one.

Map Usefulness

One of the novel features of the game, discussed in Section 4.1.4, was for the map generator to display all those areas on the map that the player had seen as opposed to have been in. So as not to influence participants in anyway it was decided not to inform users about this feature when running the user evaluation. To ascertain the worth of the feature participants were asked to rank how useful they found the map. The number of the times the map was referenced throughout the game was also recorded. These results were graphed and are shown Figure 5.8.

A related t-test of the difference in the number map references using Mouse/Keyboard (mean = 10.1) versus using Gaze/Voice (mean = 9.8) showed that the difference was not statistically significant ($T\ (df = 7) = 0.08$, $P > 0.1$, two tailed). Participants' perception of how useful they found the map was tested using a two-tailed Wilcoxon Signed-Ranks test. It showed that participants did not perceive the map to be any more useful when using Gaze/Voice (median = 6) than using Mouse/Keyboard (median = 6), ($T = 9$, $N = 7$, $P > 0.2$). So participants did not find the map to be any more useful or use it more in either case. Perhaps in a

Participant Comments

A lot of comments related to the game's collision response. One participant said "I got stuck a lot which was easier to get out of using the keyboard rather than the gaze" and another saying "I felt I got trapped sometimes near the walls." A quick fix for the poor collision response had been to stop any motion commands. So if a player got stuck

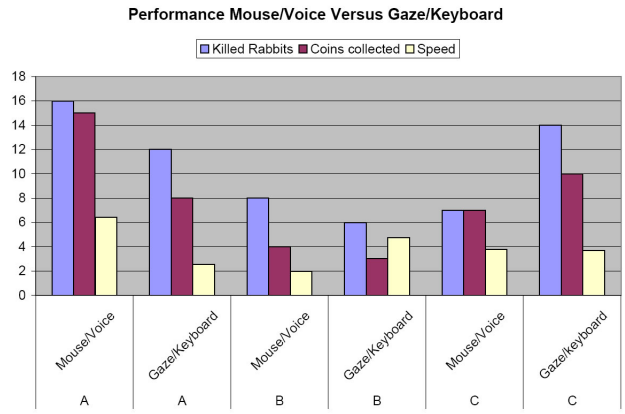

Figure 5.9: Graph of performance measures. Shows performance of Mouse/Voice versus Gaze/Keyboard in terms of rabbis shot, coins collected and speed maze was traversed.

colliding with a wall the walking or running motion would be automatically stopped. One participant commented on this "I found it annoying that the motion stopped a while after I told it to walk. I felt I had to say walk too many times."

The fact that a smoothing filter had not been applied to the gaze data was commented on by one participant who said it "was difficult to control as the cross hair jumped around the screen too much."

Voice recognition was also commented upon. One participant said it "took a while to get used to since the responsiveness from the voice input was a bit slower than mouse and keyboard" and another saying "using the voice commands felt slower than pressing a key."

One participant who found the gaze aiming helpful commented that "the gaze worked well at targeting the rabbits to be shot". Another commented favourably on the gaze camera saying "I really liked the rotation movement (left/right) of the gaze camera. It felt natural to look left when I wanted to go left, was nice, could be useful in games." A participant commented on Gaze/Voice input in general saying "experiment felt weird but was interesting."

5.2.2 Mouse/Voice Versus Gaze/Keyboard

Since only three participants took part in this study it is not possible to analyse the results in a statistically meaningful way. Figure 5.9 shows performance of Mouse/Voice

versus Gaze/Keyboard. In related t-tests looking at the differences between rabbits killed (T ($df = 2$) = 0.099, $P > 0.1$, two tailed), coins collected (T ($df = 2$) = 0.574, $P > 0.1$, two tailed) or speed (T ($df = 2$) = 0.04, $P > 0.1$, two tailed) no statistically significant difference was found. Perhaps if more participants had been involved in the study a more definitive trend may have occurred.

When asked which type was easier and which was more enjoyable two of the three opted for Mouse/Voice. Interestingly the only participant of the group to have had experience of eye trackers in the past deemed Gaze/Keyboard to be both easier and more enjoyable.

5.3 Appraisal of Results

From the results it is clear that mouse and keyboard performed better than gaze and voice. The primary problem seems to have been difficulty in navigation. The collision response appears to be the main problem in this regard. Had there been time to code the collision response so that the colliding player would smoothly travel along the wall rather than stop entirely this would not have been such a major issue. Other problems noted by users included the delay between issuing a voice command and the game reacting to this command. This is one of the facets of voice recognition and it is difficult to see how this could have been avoided in this game at least.

Despite the negative effect these problems had on players ability to navigate around the game easily, 75 % of the participants found gaze and voice the more enjoyable form of interaction. There was also a significant difference in how immersed the participants felt playing with gaze and voice as oppsed to mouse and keyboard.

Chapter 6

Conclusions & Future Work

This dissertation set out to create a game which could be controlled using gaze and voice, build a framework with which it could be evaluated and to run a user trial. The framework was implemented as a scalable tool and having been designed in an object orientated way it could easily be used with many other types of games and even different input types.

The game itself was implemented but not without its problems as discussed in Sections 4.3 and 5.3. The greatest problem being a poor collision response which could have been avoided had it been discovered earlier by means of a pilot study. It is difficult to say whether or not gaze and voice could compete with mouse and keyboard as an input modality with the results achieved. Although others in the past, such as Leyba and Malcolm [29], have reported similarly disappointing results when comparing gaze against more common input mechanisms, gaze and voice would have assuredly done better had these collision problems been resolved in a more satisfactory manner.

However some promising results were achieved. The gaze and voice option was selected by most participants as the more enjoyable and more immersive form of interaction in the study. This is a common trend of studies involving gaze based interaction in video games. Participants in trials run by Jönsson [32] and Smith and Graham [33] reported similar positive feelings towards gaze interaction. Perhaps it is only a novelty factor but it does show that people are interested in this alternative form of computer interaction.

There were no game sounds used in the created game and the user evaluation took

place in a sound proof room. This was to counteract any ill effects background noise may have had by creating false positive recognitions. This is obviously not an ideal gaming scenario. Further work could look at the effects of background noise on voice recognition in games and how to overcome it.

Most previous studies looking at gaze interaction, including Jönsson [32], Smith and Graham [33], Dorr *et al.* [36], have adapted open source games to work with gaze data. A few exceptions include Isokoski and Martin [34] and Castellina and Corno [37] who created game scenarios to compare different input types. Although adapting open source games reduces the implementation time required, these games were originally created with mouse and keyboard in mind. So the adapted game is restricted to using gaze as a replacement for the mouse rather than an input device in its own right. The gaze input acting only as a *mouse emulator*.

When a game is developed from scratch it should be free of this restriction. Unfortunately this was not the case in this study. Perhaps the goal of comparing gaze and voice directly against mouse and keyboard unwittingly manifested this idea on the project. A further study could exploit the way we use gaze and voice in the real world to create a novel, fun video game. Rather than comparing directly against mouse and keyboard the game experience itself could be measured. This could be done using a game experience questionnaire such as described by IJsselsteijn *et al.* [51].

Another area that might be worth investigating could be how the animation and AI of game characters could be adapted to react to the player's gaze and voice. More socially realistic scenarios could be created if game characters responded in appropriate ways to the tone of the player's voice and/or the focus of their gaze.

Appendix A

User Trial Questionnaires

A.1 Background Questionnaire

Please, tick the appropriate boxes:

Gender: Male ☐ Female ☐

Age: _____

Profession: _____

Have you or anyone in your family suffered from epilepsy?
Yes ☐ No ☐

How would you describe your vision?
Normal ☐ Corrected-to-Normal ☐ Contact lenses ☐ Glasses ☐

Do you have any other eye deficiencies?

_____

Have you ever used an eye tracker before?
Yes ☐ No ☐

How would you rate your expertise in playing computer games?
Give a rating between 1 and 7. 1 being very poor and 7 being very good
1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐

How often do you play video games?
Never ☐ A few times a year ☐ Every month ☐ Every week ☐ Daily ☐

Which type of game do you most like to play?
FPS ☐ Role Playing ☐ Sports ☐ Racing ☐ Other ☐
If other please state: _____

Which platform do you use to play games on?
Do not ☐ PC/Mac ☐ Wii ☐ XBox ☐ PlayStation ☐ Other ☐
If other please state: _____

What kinds of input devices have you used?
Mouse ☐ Joystick ☐ Gamepad ☐ Other ☐ Do not play ☐
If other please state: _____

A.2 Post-User Trial Questionnaire

Please, tick the appropriate boxes:

What type of interaction was used?

Mouse/Keyboard ☐ Gaze/Voice ☐ Mouse/Voice ☐ Gaze/Keyboard ☐

How easy or difficult did you find menu navigation?

Give a rating between 1 and 7. 1 being very difficult and 7 being very easy.

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐

How easy or difficult did you find moving through the game?

Give a rating between 1 and 7. 1 being very difficult and 7 being very easy.

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐

How easy or difficult was it to collect coins?

Give a rating between 1 and 7. 1 being very difficult and 7 being very easy.

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐

How easy or difficult was it to shoot rabbits?

Give a rating between 1 and 7. 1 being very difficult and 7 being very easy.

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐

How useful was the game map in completing the game?

Give a rating between 1 and 7. 1 being very useless and 7 being very useful.

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐

How well do you think you played?

Give a rating between 1 and 7. 1 being very poor and 7 being very good.

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐

How much control did you feel you had?

Give a rating between 1 and 7. 1 being very little control and 7 being a lot of control.

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐

How natural did the controls feel?

Give a rating between 1 and 7. 1 being very unnatural and 7 being very natural.

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐

How difficult did you find the game?

Give a rating between 1 and 7. 1 being very difficult and 7 being very easy.

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐

How fast did you play the game?

Give a rating between 1 and 7. 1 being very slow and 7 being very fast.

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐

How precise did the game react to your controls?

Give a rating between 1 and 7. 1 being not precise at all and 7 being very precise.

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐

How much did you enjoy the game?

Give a rating between 1 and 7. 1 being not at all and 7 being very much.

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐

How much effort was it to play the game?

Give a rating between 1 and 7. 1 being not much effort at all and 7 being a lot of effort.

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐

How immersive was the game?

Give a rating between 1 and 7. 1 being not very immersive and 7 being very immersive.

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐

A.3 Comparison Questionnaire

Please, tick the appropriate boxes:

Which interaction type was easier to use?
Game in experiment one ☐ Game in experiment two ☐

Which interaction type was more enjoyable to use?
Game in experiment one ☐ Game in experiment two ☐

Any additional comments:

Appendix B

User Evaluation Game Instructions

B.1 Mouse & Keyboard Instructions

B.1.1 Game Menu

To navigate through the menu (shown in Figure B.1) move the cursor over the menu button you wish to select and click **Enter** on the keyboard. For the user trial you will be asked to practice first using the demo version of the game, which you may do as long as you wish. Once you are happy with the controls the trial will begin.

- To play the demo game, select "*Demo*" and then "*Mouse & Keyboard*".
- To play the game and start the user trial proper, select "*Play*" and then "*Mouse & Keyboard*".

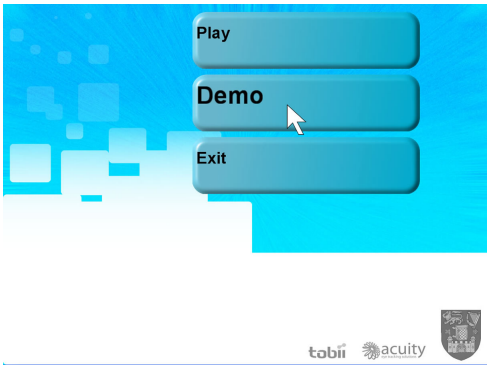

Figure B.1: Menu screen.

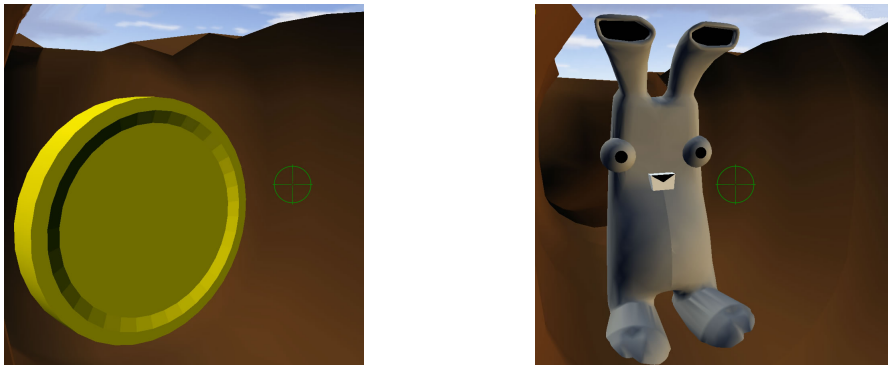

Figure B.2: Coins to be collected and rabbits to be shot.

B.1.2 Game

Objective

The objective of the game is to navigate your way through a maze and find the exit in the shortest time possible. You should also collect as many coins and shoot as many rabbits as you possibly can on your way to the exit. You will receive 10 points for collecting a coin and 10 points for shooting a rabbit. There is a time limit of eight minutes for the game.

Navigation

To change the view you are facing move your mouse around. Move it left to shift the view to the left, move it right to shift the view to the right, move it up to shift the view upwards and so on.

To navigate around the maze you can use the arrow keys to move forwards, backwards, left or right at a walking pace. To increase your speed to a run you can hold down the **Shift** key.

To collect a coin move to its general vicinity and it will be automatically collected and the points awarded to you. To shoot a rabbit aim the cross hairs at him and click the **Enter** key. Coins to be collected and rabbits to be shot are shown in Figure B.2.

To help you navigate around the maze a little easier there is a map you can bring up. The map shown in Figure B.3 will show you were you have been. The red arrow indicates where you currently are and where you are facing and the green bar indicates

Figure B.3: Shows (left) map displayed on screen and (right) maze exit.

where the exit is located. The map will only show the places in the maze you have been. So unless you have found the exit it will not appear on the map. To bring up the map press the **M** key on the keyboard.

The exit of the maze is shown in Figure B.3. It will automatically open when you stand near it. Once the door opens the game is over. If you want to pause the game at any stage, click the **Esc** key. This will bring up the Pause Menu which can be navigated like any other menu.

B.1.3 Summary of Controls

Action	**Mouse**	**Keyboard**
Shift view up	Move mouse upwards	-
Shift view down	Move mouse downwards	-
Shift view left	Move mouse to the left	-
Shift view right	Move mouse to the right	-
Move forwards	-	Up arrow key
Move backwards	-	Down arrow key
Move left	-	Left arrow key
Move right	-	Right arrow key
Move faster/run	-	Hold down Shift key
To fire bullet	Aim with mouse	Enter key
To show or hide map	-	M key
To pause game	-	Esc key

Table B.1: Mouse & keyboard game commands.

Action	**Mouse**	**Keyboard**
To select a menu item	Hold mouse over menu item	Enter key

Table B.2: Mouse & keyboard menu commands.

B.2 Gaze & Voice Instructions

B.2.1 Game Menu

To navigate through the menu (shown in Figure B.4) look at the menu button you wish to select (this will move the cursor over the menu button) and say **Option** into the microphone. For the user trial you will be asked to practice first using the demo version of the game, which you may do as long as you wish. Once you are happy with the controls the trial will begin.

- To play the demo game, select "*Demo*" and then "*Gaze & Voice*".
- To play the game and start the user trial proper, select "*Play*" and then "*Gaze & Voice*".

B.2.2 Game

Objective

The objective of the game is to navigate your way through a maze and find the exit in the shortest time possible. You should also collect as many coins and shoot as many rabbit as you possibly can on your way to the exit. You will receive 10 points for collecting a coin and 10 points for shooting a rabbit. There is a time limit of eight minutes for the game.

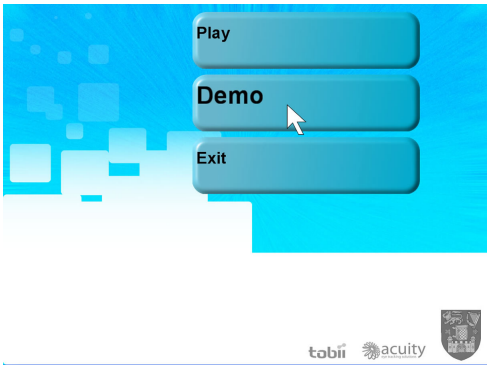

Figure B.4: Menu screen.

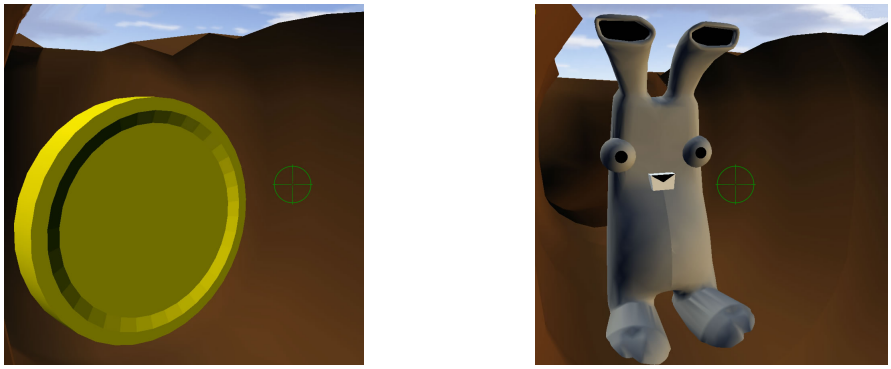

Figure B.5: Coins to be collected and rabbits to be shot.

Navigation

To change the view you are facing, move your gaze to either the left or the right of the screen. So for example if you wish to shift the view to the left, look towards the left portion of the screen - the view will gradually shift leftward. Likewise to move the view right, look towards the right portion of the screen. As means of a visual aid, an arrow will appear on screen indicating the direction the view is shifting toward, as shown in Figure B.6.

To navigate say **Walk** into the microphone. This will move you forwards in the direction you are currently facing. If you wish to move more quickly use the voice command **Run**. To halt either walking or running movement use the voice command **Stop**.

To collect a coin move to its general vicinity and it will be automatically collected

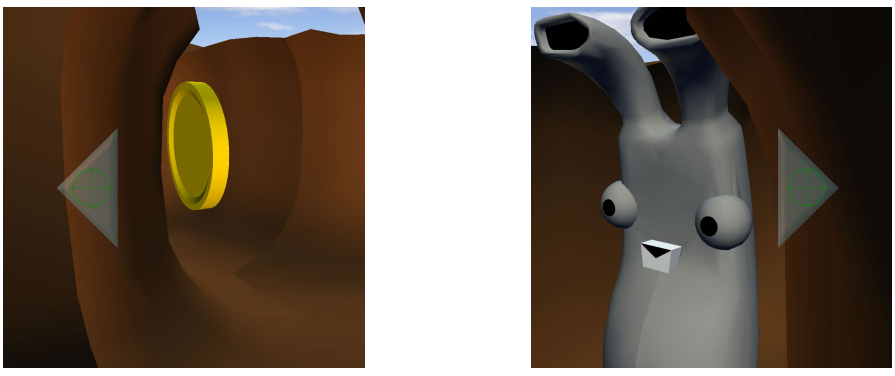

Figure B.6: Moving the view with your gaze.

Figure B.7: Shows (left) map displayed on screen and (right) maze exit.

and the points awarded to you. To shoot a rabbit aim the cross hairs at him (by looking at him) and say **Fire** into the microphone. Coins to be collected and rabbits to be shot are shown in Figure B.5.

To help you navigate around the maze a little easier there is a map you can bring up. The map shown in Figure B.7 will show you were you have been. The red arrow indicates where you currently are and where you are facing and the green bar indicates where the exit is located. The map will only show the places in the maze you have been. So unless you have found the exit it will not appear on the map. To bring up the map say **Maze** into the microphone.

The exit of the maze is shown in figure B.7. It will automatically open when you stand near it. Once the door opens the game is over. If you want to pause the game at any stage, say **Pause Menu** into the microphone. This will bring up the Pause Menu which can be navigated like any other menu.

B.2.3 Summary of Controls

Action	**Gaze**	**Voice Command**
Shift view left	Look towards the left of the screen	-
Shift view right	Look towards the right of the screen	-
Move	-	Say "**Walk**"
Move faster	-	Say "**Run**"
Stop moving	-	Say "**Stop**"
To fire bullet	Aim with gaze	Say "**Fire**"
To show or hide map	-	Say "**Maze**"
To pause game	-	Say "**Pause Menu**"

Table B.3: Gaze & voice game commands.

Action	**Gaze**	**Voice Command**
To select a menu item	Look at menu item	Say "**Option**"

Table B.4: Gaze & voice menu commands.

B.3 Gaze & Keyboard Instructions

B.3.1 Game Menu

To navigate through the menu (shown in Figure B.4) look at the menu button you wish to select (this will move the cursor over the menu button) and click **Enter** on the keyboard. For the user trial you will be asked to practice first using the demo version of the game, which you may do as long as you wish. Once you are happy with the controls the trial will begin.

- To play the demo game, select "*Demo*" and then "*Gaze & Keyboard*".
- To play the game and start the user trial proper, select "*Play*" and then "*Gaze & Keyboard*".

B.3.2 Game

Objective

The objective of the game is to navigate your way through a maze and find the exit in the shortest time possible. You should also collect as many coins and shoot as many rabbits as you possibly can on your way to the exit. You will receive 10 points for collecting a coin and 10 points for shooting a rabbit. There is a time limit of eight minutes for the game.

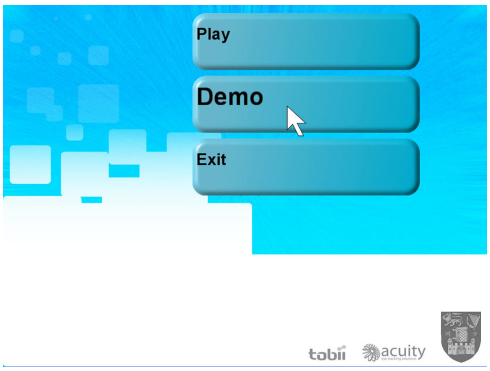

Figure B.8: Menu screen.

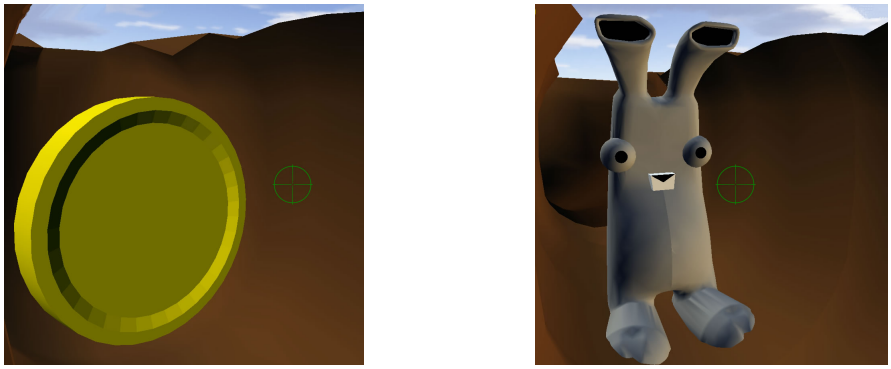

Figure B.9: Coins to be collected and rabbits to be shot.

Navigation

To change the view you are facing, move your gaze to either the left or the right of the screen. So for example if you wish to shift the view to the left, look towards the left portion of the screen - the view will gradually shift leftward. Likewise to move the view right, look towards the right portion of the screen. As means of a visual aid, an arrow will appear on screen indicating the direction the view is shifting toward, as shown in Figure B.10.

To navigate around the maze you can use the arrow keys to move forwards, backwards, left or right at a walking pace. To increase your speed to a run you can hold down the **Shift** key.

To collect a coin move to its general vicinity and it will be automatically collected and the points awarded to you. To shoot a rabbit aim the cross hairs at him and click

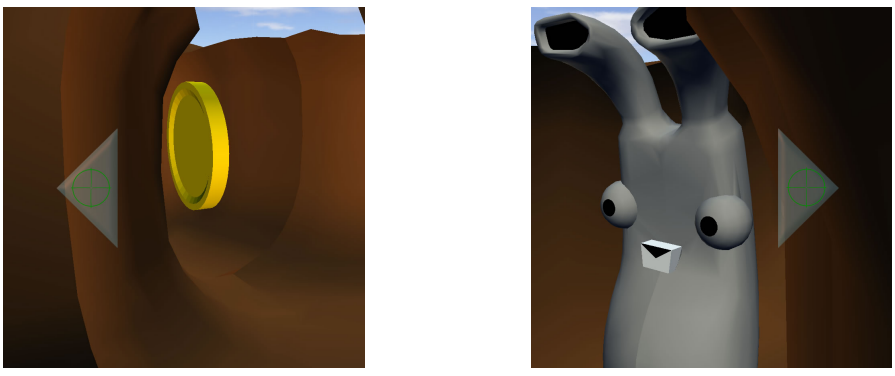

Figure B.10: Moving the view with your gaze.

Figure B.11: Shows (left) map displayed on screen and (right) maze exit.

the **Enter** key. Coins to be collected and rabbits to be shot are shown in Figure B.9.

To help you navigate around the maze a little easier there is a map you can bring up. The map shown in Figure B.11 will show you were you have been. The red arrow indicates where you currently are and where you are facing and the green bar indicates where the exit is located. The map will only show the places in the maze you have been. So unless you have found the exit it will not appear on the map. To bring up the map press the **M** key on the keyboard.

The exit of the maze is shown in figure B.11. It will automatically open when you stand near it. Once the door opens the game is over. If you want to pause the game at any stage, click the **Esc** key. This will bring up the Pause Menu which can be navigated like any other menu.

B.3.3 Summary of Controls

Action	**Gaze**	**Keyboard**
Shift view left	Look towards the left of the screen	-
Shift view right	Look towards the right of the screen	-
Move forwards	-	Up arrow key
Move backwards	-	Down arrow key
Move left	-	Left arrow key
Move right	-	Right arrow key
Move faster/run	-	Hold down Shift key
To fire bullet	Aim with gaze	Enter key
To show or hide map	-	M key
To pause game	-	Esc key

Table B.5: Gaze & keyboard game commands.

Action	**Mouse**	**Keyboard**
To select a menu item	Look at menu item	Enter key

Table B.6: Gaze & keyboard menu commands.

B.4 Mouse & Voice Instructions

B.4.1 Game Menu

To navigate through the menu (shown in Figure B.8) move the cursor over the menu button you wish to select and say **Option** into the microphone. For the user trial you will be asked to practice first using the demo version of the game, which you may do as long as you wish. Once you are happy with the controls the trial will begin.

- To play the demo game, select "*Demo*" and then "*Mouse & Voice*".
- To play the game and start the user trial proper, select "*Play*" and then "*Mouse & Voice*".

B.4.2 Game

Objective

The objective of the game is to navigate your way through a maze and find the exit in the shortest time possible. You should also collect as many coins and shoot as many rabbits as you possibly can on your way to the exit. You will receive 10 points for collecting a coin and 10 points for shooting a rabbit. There is a time limit of eight minutes for the game.

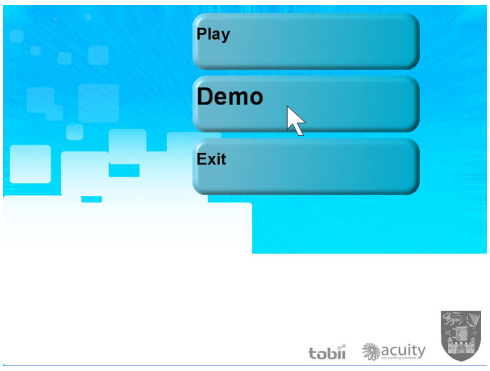

Figure B.12: Menu screen.

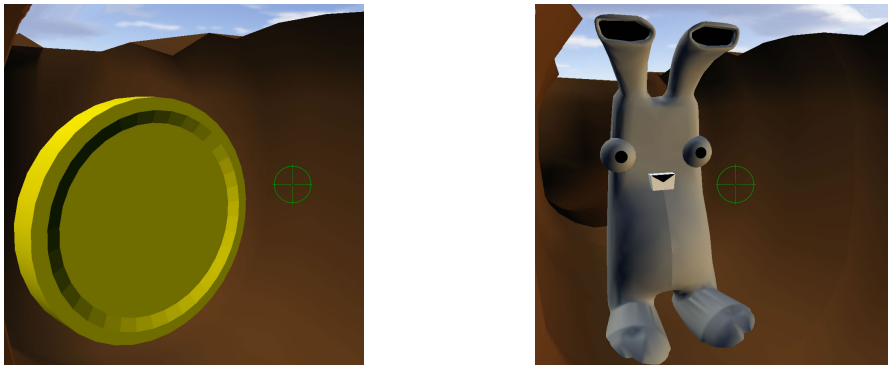

Figure B.13: Coins to be collected and rabbits to be shot.

Navigation

To change the view you are facing move your mouse around. Move it left to shift the view to the left, move it right to shift the view to the right, move it up to shift the view upwards and so on.

To navigate say **Walk** into the microphone. This will move you forwards in the direction you are currently facing. If you wish to move more quickly use the voice command **Run**. To halt either walking or running movement use the voice command **Stop**.

To collect a coin move to its general vicinity and it will be automatically collected and the points awarded to you. To shoot a rabbit aim the cross hairs at him (by looking at him) and say **Fire** into the microphone. Coins to be collected and rabbits to be shot are shown in Figure B.13.

To help you navigate around the maze a little easier there is a map you can bring up. The map shown in Figure B.14 will show you were you have been. The red arrow indicates where you currently are and where you are facing and the green bar indicates where the exit is located. The map will only show the places in the maze you have been. So unless you have found the exit it will not appear on the map. To bring up the map say **Maze** into the microphone.

The exit of the maze is shown in figure B.14. It will automatically open when you stand near it. Once the door opens the game is over. If you want to pause the game at any stage, say **Pause Menu** into the microphone. This will bring up the Pause Menu which can be navigated like any other menu.

Figure B.14: Shows (left) map displayed on screen and (right) maze exit.

B.4.3 Summary of Controls

Action	**Mouse**	**Voice Command**
Shift view up	Move mouse upwards	-
Shift view down	Move mouse downwards	-
Shift view left	Move mouse to the left	-
Shift view right	Move mouse to the right	-
Move	-	Say "**Walk**"
Move faster	-	Say "**Run**"
Stop Moving	-	Say "**Stop**"
To fire bullet	Aim with mouse	Say "**Fire**"
To show or hide map	-	Say "**Maze**"
To pause game	-	Say "**Pause Menu**"

Table B.7: Mouse & voice game commands.

Action	**Mouse**	**Voice Command**
To select a menu item	Hold mouse over menu item	Say "**Option**"

Table B.8: Mouse & voice menu commands.

Appendix C

User Trial Data

Participant	1	2	3	4	5	6	7	8
Gender	Female	Male	Male	Male	Male	Male	Male	Male
Age	30	22	30	30	26	33	23	23
Profession	Lecturer	Student	PhD Student	PhD Student	Student	Lecturer	Student	Student
Previously used eye tracker	Yes	No	No	Yes	No	Yes	No	No
Gaming expertise	4	7	7	5	6	5	5	6
How often	Monthly	Daily	Weekly	Weekly	Weekly	Weekly	A few times a year	Daily

Table C.1: Participant background data for mouse/keyboard versus gaze/voice comparison.

Participant	1		2		3		4	
Input type	Mouse/Keyboard	Gaze/Voice	Mouse/Keyboard	Gaze/Voice	Mouse/Keyboard	Gaze/Voice	Mouse/Keyboard	Gaze/Voice
Killed rabbits	11	4	8	7	9	7	11	8
Shots taken	13	6	13	9	14	11	28	12
Coins collected	11	6	5	2	8	6	11	10
Map references	19	10	4	3	15	8	13	9
Distance	141	92	74	45	165	58	149	92
Exit found	Yes	No	Yes	Yes	Yes	Yes	Yes	No
Score	220	100	130	90	170	130	220	180
Time ms	353248	480013	85541	264877	239666	284161	274601	480012
Participant	5		6		7		8	
Input type	Mouse/Keyboard	Gaze/Voice	Mouse/Keyboard	Gaze/Voice	Mouse/Keyboard	Gaze/Voice	Mouse/Keyboard	Gaze/Voice
Killed rabbits	10	6	6	8	16	7	9	6
Shots taken	11	13	7	14	25	12	10	8
Coins collected	8	5	7	6	16	5	7	5
Map references	7	22	5	14	13	4	5	9
Distance	98	70	86	71	198	56	69	102
Exit found	Yes	No	Yes	Yes	No	No	Yes	No
Score	180	110	130	140	320	120	160	110
Time ms	145450	480007	170249	401836	428199	480009	130823	480000

Table C.2: Participant recorded game data for mouse/keyboard versus gaze/voice comparison.

Participant	1		2		3		4	
Input Type	Mouse/Keyboard	Gaze/Voice	Mouse/Keyboard	Gaze/Voice	Mouse/Keyboard	Gaze/Voice	Mouse/Keyboard	Gaze/Voice
Menu Navigation Ease (1-7, difficult-easy)	7	3	7	6	7	6	6	6
Game Navigation Ease (1-7, difficult-easy)	6	3	7	6	6	5	4	3
Coin Collection Ease (1-7, difficult-easy)	7	4	7	7	7	6	5	4
Rabbit Shooting (1-7, difficult-easy)	5	6	7	5	7	6	6	5
Map Usefulness (1-7, useless-useful)	7	6	7	7	7	6	4	7
How well played (1-7, poor-well)	5	3	7	5	6	5	4	1
How much control did you have (1-7, little control - lots of control)	6	3	7	7	7	5	5	4
Naturalness of Control (1-7, unnatural - very natural)	7	5	6	6	6	5	5	3
Game difficulty (1-7, difficult-easy)	6	4	6	6	7	6	5	3
How fast did you play (1-7, slow-fast)	6	2	7	4	5	6	5	3
Control Precision (1-7, imprecise-very precise)	5	2	7	6	7	5	4	3
Game Enjoyment (1-7, unenjoyable - enjoyable)	5	4	7	6	6	6	4	3
Effort to play (1-7, little effort - lots of effort)	4	5	1	1	1	2	3	6
Immersive (1-7, not immersive - very immersive)	5	6	4	6	4	5	4	6
Which was easier	Mouse/Keyboard		Mouse/Keyboard		Mouse/Keyboard		Mouse/Keyboard	
Which was more enjoyable	Gaze/Voice		Gaze/Voice		Gaze/Voice		Mouse/Keyboard	
Participant	5		6		7		8	
Input Type	Mouse/Keyboard	Gaze/Voice	Mouse/Keyboard	Gaze/Voice	Mouse/Keyboard	Gaze/Voice	Mouse/Keyboard	Gaze/Voice
Menu Navigation Ease (1-7, difficult-easy)	7	6	6	5	7	7	7	6
Game Navigation Ease (1-7, difficult-easy)	7	4	5	3	6	6	6	4
Coin Collection Ease (1-7, difficult-easy)	6	5	5	3	7	5	5	3
Rabbit Shooting (1-7, difficult-easy)	6	7	4	4	5	7	4	6
Map Usefulness (1-7, useless-useful)	6	5	4	5	6	7	3	6
How well played (1-7, poor-well)	7	3	5	3	6	3	5	3
How much control did you have (1-7, little control - lots of control)	6	4	5	3	6	5	5	5
Naturalness of Control (1-7, unnatural - very natural)	6	4	5	2	6	5	4	4
Game difficulty (1-7, difficult-easy)	7	5	6	3	2	3	7	4
How fast did you play (1-7, slow-fast)	6	3	6	2	6	2	6	3
Control Precision (1-7, imprecise-very precise)	7	3	6	3	6	4	4	5
Game Enjoyment (1-7, unenjoyable - enjoyable)	6	6	3	2	3	4	6	5
Effort to play (1-7, little effort - lots of effort)	1	3	2	6	3	5	3	5
Immersive (1-7, not immersive - very immersive)	4	5	5	4	3	6	3	6
Which was easier	Mouse/Keyboard		Mouse/Keyboard		Mouse/Keyboard		Mouse/Keyboard	
Which was more enjoyable	Gaze/Voice		Mouse/Keyboard		Gaze/Voice		Gaze/Voice	

Table C.3: Participant questionnaire answers for mouse/keyboard versus gaze/voice comparison.

Participant	A	B	C
Gender	Male	Male	Male
Age	24	38	26
Profession	Student	Researcher	Student
Previously used eye tracker	No	Yes	No
Gaming expertise	7	1	6
How often	Daily	Never	Weekly

Table C.4: Participant background data for mouse/voice versus gaze/keyboard comparison.

Participant	A		B		C	
Input Type	Mouse/Voice	Gaze/Keyboard	Mouse/Voice	Gaze/Keyboard	Mouse/Voice	Gaze/Keyboard
Killed Rabbits	16	12	8	6	7	14
Shots taken	24	53	9	7	10	48
Coins collected	15	8	4	3	7	10
Map references	7	28	10	6	8	26
Distance	217	122	94	174	76	176
Exit found	Yes	No	No	Yes	Yes	No
Score	310	200	120	90	140	240
Time ms	339138	480010	480007	366324	200286	480004

Table C.5: Participant recorded game data for mouse/voice versus gaze/keyboard comparison.

Participant	A		B		C	
Menu Navigation Ease (1-7, difficult-easy)	6	7	2	6	7	4
Game Navigation Ease (1-7, difficult-easy)	5	3	2	6	6	5
Coin Collection Ease (1-7, difficult-easy)	5	4	4	7	5	6
Rabbit Shooting (1-7, difficult-easy)	6	3	6	7	7	6
Map Usefulness (1-7, useless-useful)	7	6	3	5	7	7
How well played (1-7, poor-well)	5	2	2	4	7	3
How much control did you have (1-7, little control - lots of control)	4	3	2	6	7	3
Naturalness of Control (1-7, unnatural - very natural)	4	3	2	6	5	4
Game difficulty (1-7, difficult-easy)	7	5	2	5	6	5
How fast did you play (1-7, slow-fast)	4	1	2	4	7	5
Control Precision (1-7, imprecise-very precise)	5	3	2	6	7	3
Game Enjoyment (1-7, unenjoyable - enjoyable)	5	3	4	6	5	3
Effort to play (1-7, little effort - lots of effort)	3	2	6	2	5	2
Immersive (1-7, not immersive - very immersive)	4	4	4	4	7	6
Which was easier	Mouse/Voice		Gaze/Keyboard		Mouse/Voice	
Which was more enjoyable	Mouse/Voice		Gaze/Keyboard		Mouse/Voice	

Table C.6: Participant questionnaire answers for mouse/voice versus gaze/keyboard comparison.

.

Bibliography

[1] M. Sanchanta, "Nintendos wii takes console lead," *Financial Times*, 2007.

[2] R. Jacob, "What You Look At Is What You Get: Eye Movement-Based Interaction Techniques," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1990.

[3] A. Duchowski, *Eye Tracking Methodology, Theory and Practice*. Springer, second ed., 2007.

[4] L. Hermann, "Eine Erscheinung simultanen Contrastes," *Pflügers Archiv European Journal of Physiology*, vol. 3, no. 1, pp. 13–15, 1870.

[5] Wikipedia, "Grid Illusion." Accessed 28 August 2009 at. `http://en.wikipedia.org/wiki/Grid_illusion`.

[6] T. Eye Health, "Eye Anatomy: Parts of The Eye." Accessed 28 August 2009 at. `http://www.thirdeyehealth.com/eye-anatomy.html`.

[7] R. Haber and M. Hershenson, *The Psychology of Visual Perception*. Holt Rinehart and Winston Inc., New York, 1973.

[8] D. Robinson, "The Oculomotor Control System: A Review," in *Proceedings of IEEE*, pp. 1032 – 1049, 1968.

[9] W. L. Shebilske and D. Fisher, "Understanding Extended Discourse Through the Eyes: How and Why," 1983.

[10] R. Leigh and D. Zee, *The Neurology of Eye Movements*. F. A. Davis, Philadelphia, second ed., 1991.

[11] D. Hubel, "Eye, Brain, and Vision," in *Scientific American Library*, 1988.

[12] R. H. S. Carpenter, *Movements of the Eyes*. Pion, London, 1977.

[13] D. E. Irwin, "Visual Memory Within and Across Fixations," 1992.

[14] Skalar, "Scleral Search Coils and Accessories." Accessed 28 August 2009 at. `http://www.skalar.nl/index2.html`.

[15] Metrovision, "Dynamic EOG." Accessed 28 August 2009 at. `http://www.metrovision.fr/mv-eo-notice-us.html`.

[16] Tobii, "Tobii X60 & X120 Eye Trackers Product Leaflet," 2009.

[17] SR-Research, "Headmount Angle." Accessed 28 August 2009 at. `http://sr-research.com/images/headmount_angle_sml.jpg`.

[18] I. Starker and R. Bolt, "A Gaze-Responsive Self-Disclosing Display," in *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems*, (New York, NY, USA), pp. 3–10, ACM, 1990.

[19] S. Hillaire, A. Lécuyer, R. Cozot, and G. Casiez, "Depth-of-Field Blur Effects for First-Person Navigation in Virtual Environments," *IEEE Computer Graphics and Applications*, vol. 28, no. 6, pp. 47–55, 2008.

[20] A. Duchowski, N. Cournia, and H. Murphy, "Gaze-Contingent Displays: A Review," in *CyberPsychology & Behavior*, pp. 621–634, 2004.

[21] C. O'Sullivan and J. Dingliana, "Collisions and Perception," *ACM Trans. Graph.*, vol. 20, no. 3, pp. 151–168, 2001.

[22] C. O'Sullivan, J. Dingliana, and S. Howlett, "Eye-movements and Interactive Graphics," *The Mind's Eyes: Cognitive and Applied Aspects of Eye Movement Research*, 2002.

[23] Q. Mehdi, X. Zeng, and N. Gough, "An interactive speech interface for virtual characters in dynamic environments," 2004.

[24] J. Larson, *VoiceXML: Introduction to Developing Speech Applications*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2002.

[25] SpeechTechMag, "Speech Technologies Make Video Games Complete." Accessed 28 August 2009 at. `http://www.speechtechmag.com/Articles/ReadArticle.aspx?ArticleID=29432`.

[26] F. Laramee, "Speech Recognition." Accessed 31 August 2009 at. `http://www.gignews.com/fdlspeech1.htm`.

[27] P. Hämäläinen, T. Mäki-Patola, V. Pulkki, and M. Airas, "Musical computer games played by singing," in *Proc. 7 th Int. Conf. on Digital Audio Effects (DAFx04), Naples*, 2004.

[28] P. Isokoski, M. Joos, O. Spakov, and B. Martin, "Gaze controlled games," *Universal Access in the Information Society*, 2009.

[29] J. Leyba and J. Malcolm, "Eye Tracking as an Aiming Device in a Computer Game," in *Course work (CPSC 412/612 Eye Tracking Methodology and Applications by A.Duchowski)*, Clemson University, 2004.

[30] S. Zhai, C. Morimoto, and S. Ihde, "Manual and gaze input cascaded (magic) pointing," in *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, (New York, NY, USA), pp. 246–253, ACM, 1999.

[31] A. Kenny, H. Koesling, D. Delaney, S. McLoone, and T. Ward, "A Preliminary Investigation into Eye Gaze Data in a First Person Shooter Game," in *Proceedings of the 19th European Conference on Modelling and Simulation*, 2005.

[32] E. Jönsson, "If Looks Could Kill - An Evaluation of E Tracking in Computer Games," Master's thesis, KTH Royal Institute of Technology, Sweden, 2005.

[33] J. D. Smith and T. C. N. Graham, "Use of Eye Movements for Video Game Control," in *Proceedings of the 2006 ACM SIGCHI International Conference on Advancements in Computer Entertainment Technology*, 2006.

[34] P. Isokoski and B. Martin, "Eye Tracker Input in First Person Shooter Games," in *Proceedings of the 2nd COGAIN Annual Conference on Communication by Gaze Interaction: Gazing into the Future*, pp. 78–81, 2006.

[35] P. Isokoski, A. Hyrskykari, S. Kotkaluoto, and B. Martin, "Gamepad and Eye Tracker Input in FPS Games: data for the first 50 min," in *Proceedings of COGAIN*, pp. 10–15, 2007.

[36] M. Dorr, M. Böhme, T. Martinetz, and E. Brath, "Gaze beats mouse: a case study," in *Proceedings of COGAIN*, pp. 16–19, 2007.

[37] E. Castellina and F. Corno, "Multimodal Gaze Interaction in 3D Virtual Environments," in *Proceedings of the 4th COGAIN Annual Conference on Communication by Gaze Interaction, Environment and Mobility Control by Gaze*, 2008.

[38] T. Wilcox, M. Evans, C. Pearce, N. Pollard, and V. Sundstedt, "Gaze and Voice Based Game Interaction: The Revenge of the Killer Penguins," in *Proceedings of International Conference on Computer Graphics and Interactive Techniques*, 2008.

[39] Tobii, "Tobii T60 & T120 Eye Trackers Product Leaflet," 2009.

[40] Acuity-ETS, "Acuity eye tracking solutions." Accessed 30 August 2009 at. `http://www.acuity-ets.com/`.

[41] Tobii, "Tobii SDK Product Description," 2009.

[42] T. Technology, "Tobii T/X Series Eye trackers Product Description." Accessed 30 August 2009 at. `http://www.tobii.com/archive/files/17995/Tobii_TX_Series_Eye_Trackers_product_description.pdf.aspx`.

[43] MSDN, "Speech API overview." Accessed 31 August 2009 at. `http://msdn.microsoft.com/en-us/library/ms720151(VS.85).aspx`.

[44] GarageGames, "Game Development Tools and Software." Accessed 30 August 2009 at. `http://www.garagegames.com/`.

[45] Unity, "UNITY: Game Development Tool." Accessed 30 August 2009 at. http://unity3d.com/.

[46] OGRE, "OGRE - Open Source 3D Graphics Engine." Accessed 30 August 2009 at. http://www.ogre3d.org.

[47] MSDN, "COM." Accessed 31 August 2009 at. http://msdn.microsoft.com/en-us/library/aa139694.aspx.

[48] MSDN, ".NET/COM Migration and Interoperability." Accessed 31 August 2009 at. http://msdn.microsoft.com/en-us/library/ms978506.aspx.

[49] V. Sundstedt, M. Whitton, and M. Bloj, "The Whys, How Tos, and Pitfalls of User Studies (Invited Course)," in *ACM SIGGRAPH*, ACM, August 2009.

[50] R. Grootjans, *XNA 2.0 Game Programming Recipes: A Problem-Solution Approach*. Apress Berkely, CA, USA, 2008.

[51] W. IJsselsteijn, Y. de Kort, and K. Poels, "The Game Experience Questionnaire: Development of a self-report measure to assess the psychological impact of digital games," *Manuscript in preparation*, 2008.