UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

FYP REPORT

LIMITED DIMENSION

DESKTOP VR DISPLAY GAME DESIGNED WITH UNITY 3D GAME ENGINE AND MS KINECT

Name - Denis Bartley

Student Number - 09001739

FYP Supervisor – Dr. Conor Ryan

Table of Contents

ACKNOWLEDGMENTS

I would like to thank Dr. Conor Ryan for his support and guidance that made this project possible.

Johnny Lee for inspiring this project and inspiring people to hack and create something new.

And finally to my friends and family that I asked to test this game, without them some bugs would have never been seen.

ABBREVIATIONS

HCI	- Human Computer Interaction
TDD	- Test Driven Development
SDK	- Software Development Kit
AI	- Artificial Intelligence
GUI	- Graphical User Interface
API	- Application Programming Interface
2D	- Two-Dimensional
3D	- Three-Dimensional
Fig	- Figure
UV	- Denote the axis of 2D texture
LED	- Light Emitting Diode
IR	- Infrared
XP	- eXtreme Programming
SLERP	- Spherically interpolate between two vectors

GLOSSARY

Texture	- The graphic the is applied to a model
Rigidbody	- Control of an object's position through physical simulation
Platformer	- A game that the player is set on a linear playground
T Pose	- Player stands with their hands stretched out horizontally
Parallax	- The effect whereby the position or direction of an object appears to differ when viewed from different positions (Dictionary.com 2013)
Desktop VR Display	- The illusion of displaying a 3D object on a screen depending on the users perspective

Orthographic Projection – Representing 3D shape in 2D.

ABSTRACT

The goal of this project is to project is a puzzle game built using the Unity 3D engine (Unity 3D 2005) in conjunction with the Microsoft Kinect. The game is a 2D / 3D puzzle platformer that allows the player to navigate the gaming environment using both 2D and 3D perspectives of the map. The Microsoft Kinect is used to alter the camera angle depending on the player's location in front of the screen and see what cannot usually be seen in the game levels.

This project takes the traditional platform gaming experience and combines both 2D and 3D to give the player a diverse puzzle gaming experience. Three different camera modes can be activated. The first is the 2D the second is the 3D and the third is another 3D environment that allows the player to control the camera by physically moving and looking around the environment. This effect allows the user to see items that may be hidden in both views or to move the character between platforms by changing the camera angle.

The game uses tag-finding AI to find the players location and direct a projectile towards the player. All objects in the game have been designed using Blender 3D software and designed in a way that both are viewed in the 2D and 3D views.

This game has been created in a way that takes advantage of Unity's prefab system that allows other people to download the game and create their own levels or add to the project. The current build of the project is available online at http://limiteddimension.wordpress.com/

INTRODUCTION

1.1 General Information

This project is designed to take advantage of the Unity 4 Game Engine and the Microsoft Kinect. Design inspiration has been taken from games such as Looksley's Line Up (Nintendo 2010) and Echochrome (Sony 2008) combined with the Johnny Lee head-tracking (Lee 2008) concept to create a game that will be challenging and engaging to the character. "Looksley's Line Up" (Nintendo 2010) for the Nintendo DSi handheld console is one of the only games available on the market that uses facial tracking to know the location of the player's head and move the camera to give the player the illusion that they are looking into something that resembles a box. The player moves the device to see around the box room and touch the screen to find objects. Echochrome's (Sony 2008) game mechanic is based on moving a camera and depending on the camera angle the game environment responds differently. The aim of the game is to move the character from one platform to another. However the platform has holes that the player can fall through. Replacing the ability to jump the player needs to move the camera until the camera view hides the holes so from that perspective so that the hole do not exist.

(Lee 2008) In 2007 a PHD student of Carnegie Mellon University's named Johnny Lee released a video on YouTube demonstrating a Wii controller hack that tracked the location of his head in relation to the sensor. He created glasses with two UV leds attached and connecting the Bluetooth Wii controller to a PC; he could then get the x, y axis. By getting the size of the screen and distance between the LEDs he could determine how far the player was from the screen. Using this simple hack he put the sensor under a TV and then wrote a program that moved a camera in relation to his movement and location in the room while displaying a realistic 3D environment on the screen.

The game's graphics have taken inspiration from a lot of mainstream platform games such as Mario (Nintendo 1996) or games available on mobile platforms. Both 2D and 3D use the same textures but specific colours are used to express the difference between 2D and 3D objects. All the art is designed using Gimp (Gimp 1996) open source image editing software and a Wacom Sketch Pad. All in game models are made using Blender 3D (Blender 1995) modeling software.

1.2 Background and Motivation

I picked this project to demonstrate the skills and techniques that I have gained from my studies and also learn new skills. This project allowed me to learn how to take on a project from beginning to end this includes all the steps starting with design, requirements, development, digital art and testing. I have also chosen to use the Kinect in my project, as I have always been interested in the Kinect technology and also the project from Johnny Lee the desktop VR display (Lee 2008). Johnny Lee used the Wii controller to track a user's head movement and using this he could display 3D objects on the screen and from the player's perspective it simulates looking through a window.

Figure 1.1 Screenshot from the Nintendo DS game Looksley's Line Up demonstrating facial tracking.

The game "Looksley's Line Up" (Nintendo 2008) uses a Nintendo DSi to give a similar effect. Instead of using a depth perception camera to track the user's head it uses a front mounted camera on the device and uses facial recognition software to track the

user's face. Using the Kinect allows the player to get the same experience as Johnny Lee's (Lee 2008) project but without the use of an invasive head mount. This new revolutionary style of Human Computer Interaction has interested me for years and trying to implement this into a gaming environment would prove to be a challenge.

The Unity 3D engine has also been of interest to me for a while, as it is a free tool that is packed with powerful tools for developing games. I have been involved in the modification of a game called Half-Life (Valve 1998), this is a first person shooter game but I picked up skills in basic map building and texture editing.

1.3 Objectives of Proposed Work

The objectives of my project are as follows:

- Use a game engine to recreate the Johnny Lee head tracking and apply it to an original gaming concept.
- Create a playable with the Unity 3D engine
- Create a project that can be modified by the public
- Improve my development / management / planning / design skills.

1.4 Interest in the Project

After creating a prototype the author uploaded a video of the head tracking on to YouTube (Bartley 2012). After a few weeks online I received some feedback from a company called spinTOUCH (spinTOUCH 2012) inquiring if I could help with a PC version that displays 4K videos instead of a game. The company specializes in developing new forms of HCI (Human Computer Interaction) such as innovative touch screen commands. The company is looking into creating a 4K screen that acts as a window. 4K is a screen resolution of 3840 x 2160; it is a relatively new technology that has only been made affordable within the last few months. The use of a screen as a window is another relatively new idea that has spawned from Johnny Lee's (Lee 2008) project. As a person

is tracked in a room their point of view of the screen imitates a window playing recorded footage. After some research it became apparent that this might become popular in the hospitality industry such as hotel rooms that don't have real windows. I informed spinTOUCH (spinTOUCH 2012) that I'm currently researching this technology and would be interested in helping after my studies.

1.5 Target Audience

This project is aimed at both gamers and developers. The game demo is targeted at both young and old gamers that want something different. Although the puzzles can be difficult it does challenge the player and every puzzle is not as difficult once it is completed the first time. The game design allows it to be recreated and altered by anyone that is familiar with Unity 3D (Unity 2005) game engine. Documentation has been added to explain how to edit the game as well as any mechanics any developer may want to use in another project.

RESEARCH

2.1 Technology Available

2.1.1 Wii Remote

The main inspiration for this project was the technique used by Johnny Lee to create head tracking and display 3D objects on the screen. “Although the Wii remote’s official specifications are unpublished, the global hacking community has collectively reverse-engineered a significant portion of the technical information regarding its internal workings.”(Lee 2008, p.40) The remote consists of an infrared camera sensor that is located at the tip of the controller and this provides high-resolution tracking. An accelerometer manufactured by Analog Devices provides information on movement of the controller. The sensor bar uses light emitting diodes that provide feedback and tracking for the infrared sensor, this is usually placed under or on top of the TV. All information is sent back to the gaming device or in this case the PC through a Bluetooth connection. Using this technology he realized that if you mount the Wii remote under the TV and wear glasses with light emitting diodes attached the x, y coordinates, rotation and estimated distance could be detected. Knowing the physical separation of the IR emitters you can estimate the heads distance from the screen. Taking the TV physical dimension and the user’s head distance “you can simulate the behavior of a window providing motion parallax and a changing field of view, thus increasing the illusion of depth and realism.” (Lee 2008, p.43)

To imitate the Johnny Lee project a research paper from the Department of Computer Science University of Copenhagen titled “Head tracking using a Wiimote” (Hejn and Rosenkvist 2009) proved to be an insightful and precise explanation of how the effect was created. Some of the information is already known such as the LEDs being read by the Wii remote. However getting the distance of the user from the remote proved to be more complex as the Wiimote does not have a depth camera. “To calculate

this distance, the distance between the two LEDs is scaled with the size of the monitor and used to derive the relative distance between the user and the monitor. This distance is then used to calculate the position of the user relative to the monitor: By using the sine-function on the angle which indicates the distance between the LEDs on the x-axis, the x-coordinate relative to the camera-space is known." (Hejn and Rosenkvist 2009, p.8)

The Wii's remote has a limited horizontal view field that may disrupt movement tracking. Since this study new innovation in motion tracking has been released to the domestic consumer market. Such as the Kinect (Microsoft 2010) body tracking or improved facial tracking using a standard camera have been released and would allow similar effects without the use of glasses.

2.1.2 Web Camera

Unity's supported web cam software is quite limited and the program FaceApi (FaceAPI 2012) seems to be one of the best and supports an array of features. The software tracks the user's face as well as understands facial expressions by using a standard web cam. To integrate FaceAPI with Unity 3D, an install of the standard free version of the FaceAPI software is required. Then the FaceAPI package is imported through the Unity Package Import utility. Once the package is imported a stream is made to communicate the facial coordinates to Unity see Fig 1. The data in the command windows shown are the X, Y, Z position of the face, the X, Y, Z rotation of the face and the confidence of the coordinates. The confidence is how accurate the software is sure that the data collected is correct from the users face. This data can then be applied towards a camera in the game to create the Virtual Desktop Effect similar to Johnny Lee. After setting up this software it found that it did not support many cameras and when finding a supported camera the software had difficulty reading faces in poor lighting.

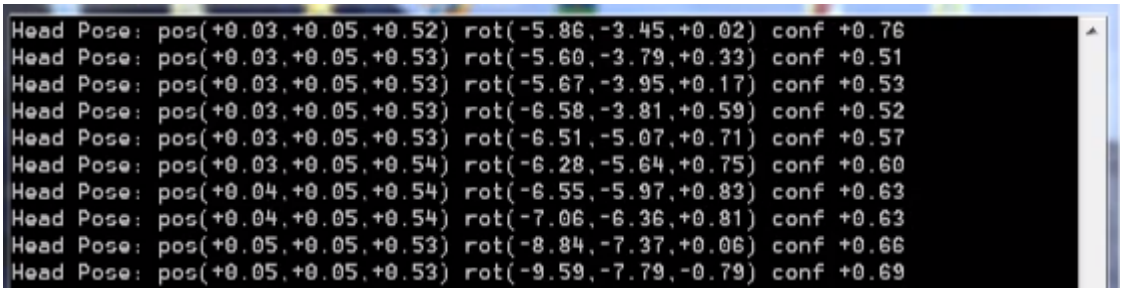

Figure 2.1 – Command line for FaceAPI

2.1.3 Microsoft Kinect

The Microsoft Kinect seems like a suitable choice for this kind of operation in a game. The Kinect is made up of an infrared projector, infrared camera, a colour camera and a four-microphone array. With these combined it provides full-body 3D motion capture, facial recognition and voice recognition capabilities. “The IR projector is an IR laser that passes through a diffraction grating and turns into a set of IR dots. The relative geometry between the IR projector and the IR camera as well as the projected IR dot pattern is known. If we can match a dot observed in an image with a dot in the projector pattern, we can then reconstruct it in 3D using triangulation.” (Zhang 2012, p.5) With the Kinect’s ability to apply skeletal tracking, head-pose and facial-expression tacking it makes this the ideal device for this project and could also be used to test character animation.

2.2 Existing Projects and Products

The Kinect has been used in quite a few projects over the previous years but one that tracks the user and plays audio within a virtual room is the thesis by Alan Holmes (Holmes 2011). It highlights the uses of different technologies available and the implementation of the Kinect for tracking. The author used the Playstation Eye to track points on the users’ head and the Kinect to track the skeleton in a virtual world. Although this technology was used with Max MSP it did provide insightful information for the setup for this project.

Since the creation of head tracking to create a VR Desktop it has remained an underground technology that hasn't had much use in the mainstream. Only one recent use is in the game Forza 4 (Turn10 2011) for the Xbox 360. In an additional feature in the game you can use the Kinect to track the players head and look around a car in real time. The effect is interesting but lacking the 3D effect that imitates realism due to lack of debt.

2.3 Selecting the Suitable Game Engine

With many free game engines available on the market, picking a suitable engine proved to be challenging as support for Kinect was a necessity. The ability to create a 2D and 3D environment in one gaming space also needed to be possible. Comparing the best free 3D game engines available and then researching the different features available for this project proved to be a difficult process. After researching Kinect plugins available for free game engines, the Unreal Engine 3 (Unreal 2012) and the Unity 3D (Unity 3D 2005) engine have the best support for the Kinect. The Unreal Engine is one of the most used engines at the moment with a lot of the mainstream games being released monthly. However the Unity engine is relatively new in the field but its library of games is growing constantly. Both engines support the "Develop Once, Publish Everywhere!" idealism. This means that once development on a game is finished it can be published to multiple platforms without any major code alteration. Both engines have a large collection of books and forums online. However Unity's community is better for helping small developers and the community is growing at a faster pace. Unity has also an affordable publishing package such as a $400 (Unity 3D 2012) one off payment for Android publishing. Unreal Engine publishing is expensive and harder to push to different platforms.

2.4 Unity Resources

To get the basics of Unity tutorials from walkerboystuido.com (WalkerBoyStuido 2012) and reading Beginning 3D Game Development with Unity (Blackman 2011) gave great insight to Unity's functionality. The author started learning Unity by creating simple objects in the engine and also learning about the different windows and tools available. Along with the Unity API (Unity 3D API 2012) and the Unity forums (Unity Forum 2012) proved to be a great source of knowledge for this project's development.

GAME DESIGN DOCUMENTATION

3.1 Game Design Document Introduction

This section of the report is a document that was created to break down the code and provide a guide on the layout of the game. The layout has been sampled from (WalkerBoyStuido 2012) tutorial with additional specific information for this project. It simplifies the projects goals and requirements as well as acting as a reference for scheduling. Design documents are living documents that change throughout the development process. In this case the original version of this document housed only the main concept and how the code will be implemented. It is now a full list of code used and an outline of the final product. This game design document is also available online to act as a reference guide for anyone that wants to use the game mechanics.

The Game Design Document is available online along with the games source code at http://limiteddimension.wordpress.com/

3.2 Game Design Document

The player needs to get the character from one side of the map to the other to complete the level. The player also controls the character in both 2 dimensional and 3 dimensional spaces to allow them to solve puzzles. The character has the ability to move certain objects such as a box to get to areas of the map.

3.3 Gameplay

The player is required to navigate from the left to the right and get to the other castle at the end of the map. The player is required to navigate through 2D and 3D environments and complete puzzles along the way. The player is limited to different actions in each view.

3.4 Player Objective

Solve puzzles and get to the other side of the level.

- Learn to Move
- Learn to Control Camera Modes
- Learn to Jump
- Push Objects
- Use the Kinect head tracking

3.5 Core Gameplay Mechanics

Player Character Abilities

- Jump
- Push
- 2D Movement
- 3D Movement

Enemy Character Abilities

- Search Radius
- Throw Weapon

3.6 Camera

This game uses both the orthographic and the perspective camera modes. Orthographic mode is used to display 2D area and perspective is used for both 3D views. The 2D and 3D camera follow the character throughout the game play and uses a zoom function to zoom into the character when not moving. The Kinect will also control one of the 3D cameras by tracking the player's head movement. A limited amount of time is enabled in the 3D view to prevent overuse. Orb placed in the map can collection the time limit.

3.7 Controls

Both the keyboard and Xbox 360 controller can be used to control the game. (Use of the Xbox controller is advised)

Player Action	Button	Keyboard
Move	Left Analog Stick	WASD
Jump	A Button	Space Bar
Long Jump	X Button	X
Push	B Button	Z
3D Camera	Left Shoulder Button	V
Kinect Camera	Right Shoulder Button	B
2D View	Y Button	C

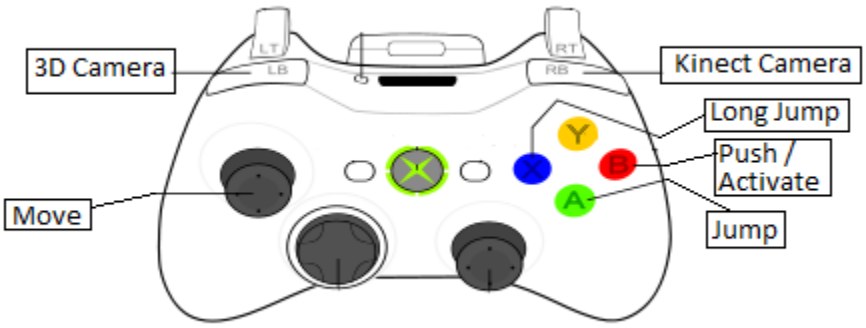

Figure 3.1 – Xbox Controller Layout (Wikipedia 2013)

3.8 Characters

Figure 3.2 – Barr (Main Character)

Figure 3.3 – Cletus (Enemy)

Figure3.4 – Follower (Enemy).

Two Characters have been designed with added animation to be used in triggered sequences *fig 3.2* and *fig 3.3*. Barr from *fig 3.2* is the main character in the game and triggers movement animation such as walk, run and jump. (These can be found in the Inspector in the Player Object)

Cletus from *fig 3.3* is triggered at a certain moment in the game and animation such as throw; run and jump are also assigned.

Follower form *fig. 3.4* watches an area around the model and once the player enters the area the model moves towards the player. If a collision is detected between the model and the player, the players' lives counter decrements.

3.9 Game Objects

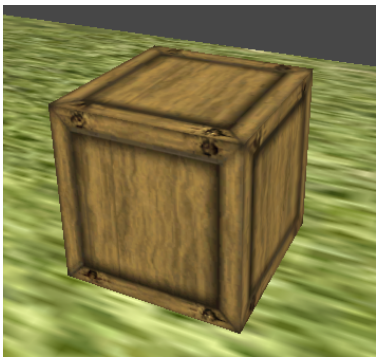

Figure 3.5 – Box (2D & 3D Control)

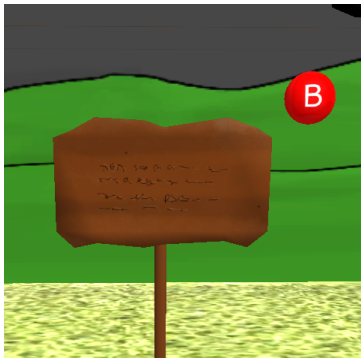

Figure 3.6 – Sign (Text Display)

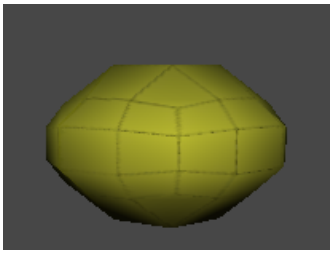

Figure 3.7 – Orb (Time Ext).

Fig. 3.5 is a box that can only be moved in the 2D camera mode. The box can be customized in the Inspector to enable the 2D and 3D modes. The 3D mode available moves the box into the background when 3D is enabled.

Fig. 3.6 is a sign for tutorial mode; this is used in the Demo to explain how the game works and can be customized by altering the path to the texture in the Inspector.

Fig. 3.7 is an orb to increase the Players time limit in 3D. Once the player collides with the object a particle effect is triggered and 2 seconds is added to the Player's time by default.

Figure 3.8 – Tree & Bush

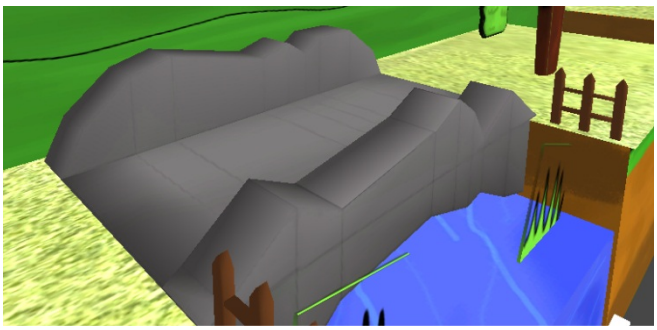

Figure 3.9 – Bridge (3D Bridge Puzzle)

Fig 3.8 is an example of trees and bushes added to the games assets catalogue to create an aesthetic looking game.

Fig 3.9 is another example of aesthetic models that have been added to the game and the bridge is also a part of an in game puzzle.

3.10 Assets

List of all classes used in the game. This document has evolved throughout development as the need for code changed.

CODE	Name	Description	Complete	File Type
AI	CletusAnim	TrigersPlayerAnimation	Yes	.cs
	ControlMatch	ControlBattleTime	Yes	.cs
	SpearAI	ControlsSpearTracking	Yes	.cs
	TrackPlayer	ReturnsPlayerCordinates	Yes	.cs
Follower	FollowerDamage	CollideWithPlayerDecrementLife	Yes	.cs
	FollowerDestroy	CollideWithPlayerDestroyObject	Yes	.cs
	FollowPlayer	MoveTowardsPlayer	Yes	.cs
Backdrop	MoveBackground	ParallaxBackground	Yes	.cs
BoxMov..	Box2D3D	MoveBoxInCameraMode	Yes	.cs
	BoxCollisionDetector	BoxMovementDirection	Yes	.cs
	WoodenBox	WoodenBoxProperties	Yes	.cs
Camera	Camera2D	2DCameraSettings	Yes	.cs
	Camera3D1	3DCameraSettings	Yes	.cs
	StationaryCam	HoldCameraPositionIn2D	Yes	.cs
	CameraListener	ControlsCameraSelection	Yes	.cs
Character..	CharAnimRot	TriggerBossAnimation	Yes	.cs
	CharController2D	2DControlProperties	Yes	.cs
	CharController3D	Generic3DControlProperties	Yes	.cs
	CharController3D1	Specific3D1ControlProperties	Yes	.cs
	CharController3D2	SpecificKinectControlProperties	Yes	.cs
	CurrentGround	ReturnsCharacterContactGround	Yes	.cs
GUI	MainMenu	MainMenuProperties	Yes	.cs
Kinect	ControlKinectFloor	Hide/DisplayEventFloor	Yes	.cs
	HeadMonitor	StoreUser'sHeadLocation	Yes	.cs
	KinectCameraContoller	ModifyKinectValuesForCamera	Yes	.cs
	ViewTrigger	TriggerEventFromHeadLocation	Yes	.cs
LevelTrig..	BossScene1	PauseControlPlayMessage	Yes	.cs
	BridgeLever	TriggerEventPlayAnimation	Yes	.cs
	ButtonEvent	TriggerEventFromButton	Yes	.cs
	DestroyParticle	DestroyParticleAfter5Seconds	Yes	.cs
	InvisWall	PlayerPassWallBecomesSolid	Yes	.cs
	KillFloor	PlayerCollisionTriggerRespawn	Yes	.cs
	RockRoll	TriggerRockRollingEvent	Yes	.cs

	RockTrigger	TriggerEventStartRockRoll	Yes	.cs
	SignCalls	DisplaySign	Yes	.cs
	SignTextDisplay	DisplayAssignedTexture	Yes	.cs
	TextDisplay	ControlTextDisplayEvent	Yes	.cs
	timeCollection	AddTimeTo3DTimeLimit	Yes	.cs
	WaterFallTrigger	PlayerTriggerEventInWaterfall	Yes	.cs
CheckPo..	CheckPointLives	CheckPointArea-Respawn	Yes	.cs
	CheckTrigger	ManuallySetCheckpoint	Yes	.cs

	Name	Description	Complete	File Type
Design	DesignDocument	ContainsInformationOnDesign	Yes	.docx
	Level1.9	DemoScene	Yes	.unity
	Menu1.0	MainMenuForGame&Prototype	Yes	.unity

	Name	Description	Complete	File Type
Art	Barr	MainCharacterTexture	Yes	.png
	Cletus	EnemyCharacter	Yes	.png
	2DTexture	Define2DObject	Yes	.png
	3DTexture	Define3DObject	Yes	.png
	clouds	CloudTextureParalax	Yes	.png
	dirt	DirtTexture	Yes	.png
	grass	GrassTexture	Yes	.png
	ground	GroundTexture	Yes	.png
	hills	HillTextureParalax	Yes	.png
	hillsHigh	Hill2TextureParalax	Yes	.png
	pillar	WallTextureKinect	Yes	.png
	sign	SignTexture	Yes	.png
	wall	WallTexture	Yes	.png
	water	WaterTexture	Yes	.png
	Wood_Box	WoodBoxTexture	Yes	.png
	target	WiiTrackingTarget	Yes	.png
	grid	WiiTrackingGrid	Yes	.png
	black	BlackTexture	Yes	.png

	Name	Description	Animated	Complete	File Type
Models	Barr	MainCharacter	Yes	Yes	.blend
	Cletus	EnemyChar	Yes	Yes	.blend
	Bridge1-1	BridgePuzzle	No	Yes	.blend
	Bridge2	DrawBridge	Yes	Yes	.blend
	Bush1	Bush	No	Yes	.blend
	buttonSign	SignForButon	No	Yes	.blend
	castleWall	LargeWall	No	Yes	.blend
	follower	FollowerEnemy	No	Yes	.blend
	gate	GateModel	No	Yes	.blend
	Rock	RockModel	No	Yes	.blend
	sign	MessageSign	Yes	Yes	.blend
	spear	SpearProjectile	No	Yes	.blend
	timeCollect	TimeExtension	Yes	Yes	.blend
	Tree1	TreeModel	No	Yes	.blend
	Tree2	TreeModel	No	Yes	.blend

3.11 GUI

The GUI consists of a main menu that demonstrates another type of view that is possible by adjusting the Kinect head tracking modifiers.

3.12 Performance of Game

This game is created with low processing and memory usage in mind. The game alone doesn't take much processing and memory so porting to a mobile platform is possible. Using the Kinect does add to the performance usage quite a bit and limits the game to PC only but could be played on low spec systems.

3.13 Redesigning the Game Notes

Certain assets within the game are made available to anyone that whishes to create their own level or use some of the game mechanics. Certain assets in the game can be dragged into the game environment such as the AI Follower. This and other assets are available in the "Prefab" folder in the Project window. Any of the Game's Objects properties can be edited in the Inspector window; Kinect head tracking can be altered in

the “cam3D2” object. Please download the Kinect DSK from MICROSOFT before building this game.

3.14 Demo Level Design

The demo is used to just demonstrate the game mechanics. It was originally designed on paper after all the game mechanics were complete. Only the boss section and the rock puzzle are considered “hard coded” but these can be applied to other maps.

UNITY DEVELOPMENT ENVIRONMENT

4.1 Unity Brief Introduction

Unity 3D is a game engine that provides the tools to create games and publish them to multiple platforms. The engine supports a vast collection of file types for graphics, sound and modeling. For instance Blender (Blender 1995) is an open source tool for creating 3D models that works hand in hand with Unity 3D. The windows for Blender and Unity 3D can be open side by side and any adjustments made in Blender are applied automatically in Unity 3D. This kind of integration can save a lot of time in design as well as encourage trying new model techniques on the fly.

Unity's code can be written in UnityScript (similar to JavaScript)(Unity 3D API 2012), Boo (similar to Python) (Codehaus 2003) and C#(Microsoft 2000). Not much research has been carried out in determining what code language is used to create the most efficient game. However experienced Unity users claimed that C# is faster as UnityScript usually includes a lot more overhead and effects the game's efficiency. Considering that the resources for the Boo coding language is not as common as other languages, using it would cause development issues for this type of project. Even though UnityScript seemed to dominate tutorials online the principles for UnityScript could be implemented towards C#. Also using C# gave the developer the opportunity to learn a new coding language. (HoBlog 2013)

The basic principle of creating a game in Unity is to create a game object within the game and apply code to that object to make it do something. Game objects can be physical or transparent and triggers can be applied to flag a collision with an object. The engine also uses a hierarchy system to group the game objects This is useful when the parent game object moves all objects once the parent moves. The hierarchy system is also useful when scripting an event that occurs through the parent and child. For instance it is used in the FollowerDestroy.cs

(DestroyObject(transform.parent.gameObject)) script that calls the parent for destruction and as this is a child of the parent it will also destroy itself.

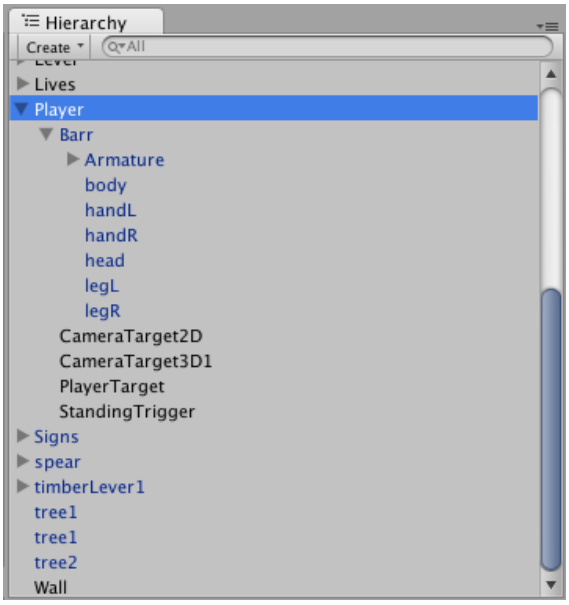

Figure 4.1 – Example of the Player Game Object Hierarchy

Using the time function in a game is one of the most important tools for creating effects and controlling events. While building this project the Time.time and Time.deltaTime were essential for solid character movement. Time.time returns the amount of time the game has been running in seconds, Time.deltaTme returns the amount of time it took to complete the last frame. The deltaTime function is used while applying movement to the player to allow for fluid motion and to keep movement consistent even during high computation.

4.2 Game Objects

Unity 3D engine has a vast range of features to help develop the game the designer wants. Unity's development environment involves coding a script for a specific purpose and then linking the code to any object in the game. All the code is written and then compiled into code that Unity understands. Existing scripts can be linked to any object in the game and give it the features from the code; this is useful as

hard coding object functions can be avoided. The IDE also has an instigator that allows viewing and editing variables on the fly, this is useful as a coder can assign variables to code with default values. If the designer needs to change any values it is done by clicking and dragging an adjuster. Using this method the designer doesn't need to look through the code to get the desired effect. Unity also takes little time to build the current project and in turn this allows rapid iteration of project development and testing. Unity also uses a tagging system that assigns a tag to any game object. This is used throughout the game as to define collision detection by reading the tag and also the AI searches for a certain tag and returns the location.

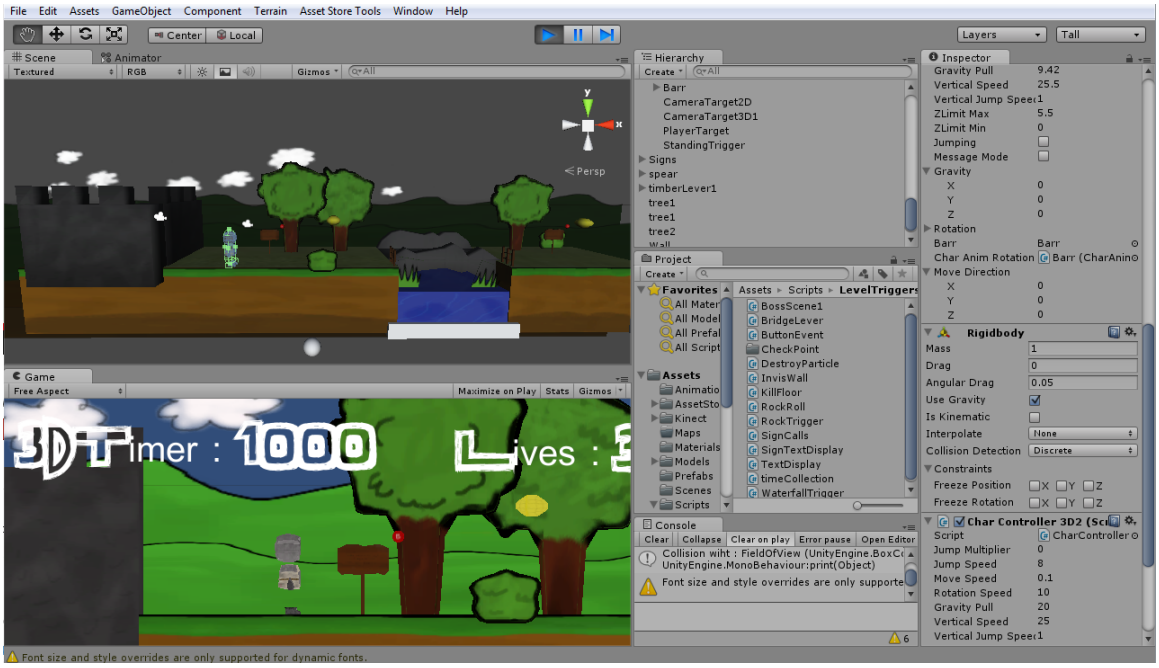

Figure 4.2 – Unity 3D Overview

Unity has several tab windows that help with developing a game and seeing the current project and its assets.

- **Scene** View *Fig 4.2*

This is where the placement of all the assets occurs similar to a blank canvas. You can move the objects around the map in this window.

Figure 4.3 – Unity 3D Scene Window

- Game View *Fig 4.3*

This is the view of the current game, once the game is run this is the view the build version will have.

Figure 4.4 – Unity 3D Game Window

- Project Window *Fig 4.4*

This is where all the assets for the game are stored this includes maps, object and code.

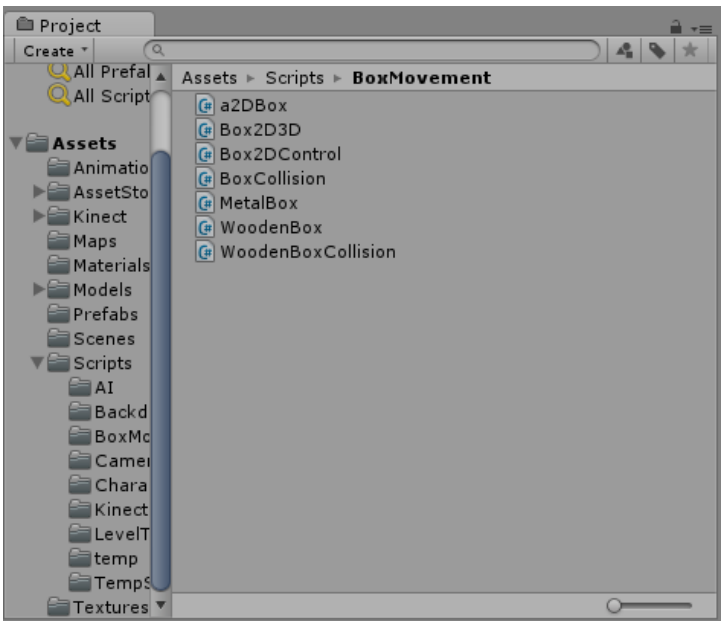

Figure 4.5 – Unity 3D Project Window

- Hierarchy Window *Fig 4.5*

This shows the immediate **scene** that is being worked on and the assets that are currently in the **scene**.

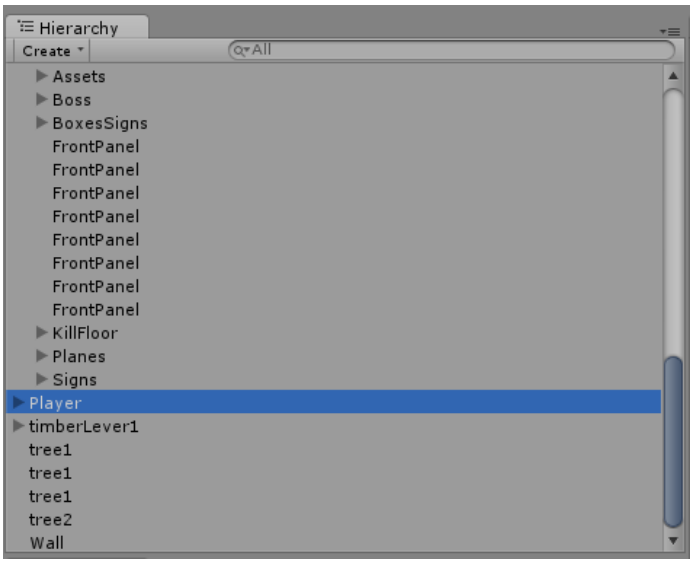

Figure 4.6 – Unity 3D Game Window

- Inspector Window *Fig 4.6*

This shows all the options and parameter available for all the items in the game.

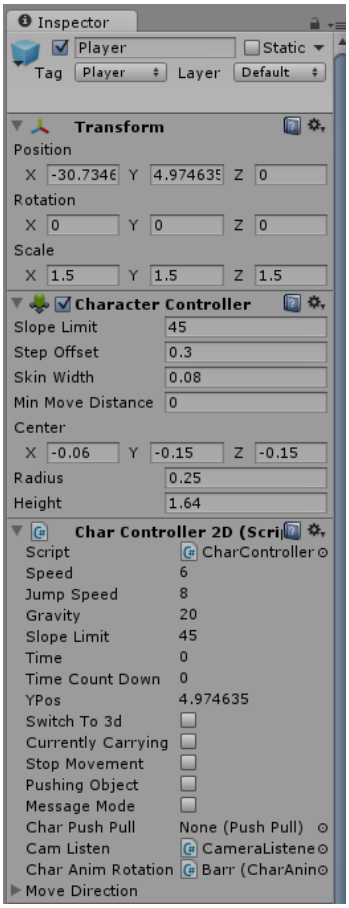

Figure 4.7 – Unity 3D Inspector Window

Another useful feature of Unity is the ability to create tools within Unity so that the game designer can use certain tools when needed. This is done by developing a certain tool for example a kill object, this is an object that if the player touches he re-spawns. The game designer can easily select a section of the map select the tool and that kill property is added easily. Many tools can be developed so the designer wouldn't need any prior knowledge of the code to create a game. The use of prefabs is also a powerful tool for re using objects in the game world. Once an object is created a prefab can be created that creates an instance of that object. This stores all variables, assets and models to one object in the project area. A designer can then click and add that object to the game at any moment.

PROTOTYPE

5.1 Original Prototype

The first prototype was created during August as a test to insure that the concept was possible and that to test if the developer felt comfortable in the development environment. The prototype was designed in JavaScript as the developer felt that it may be too time consuming learning C# as the project was not yet confirmed with the supervisor. Once a suitable prototype was created and the project was green lit by the supervisor, the developer first needed to converted from UnityScript to C#. Once a certain level of knowledge in C# was achieved this process of conversion was relatively fast and bug free.

5.2 Kinect Setup - Development

To enable the Kinect in Unity the Kinect SDK (Microsoft 2013) from Microsoft and also Visual Studios 10 need to be installed. The SDK provides sample projects from Microsoft and is useful in understanding how the Kinect code works by being able to read the information that is transferred. The Kinect wrapper (Unity Kinect Wrapper 2012) then imports the Kinect into Unity with a sample skeleton that detects the player's movements from the Kinect. The prototype "Desktop VR Display" was created by assigning a camera to the head of the Kinect skeleton and creating a map that would allow a user to demonstrate movement and camera tracking. The wrapper also has a useful emulator that can record your movements and play them back without a Kinect connected to the system.

5.3 Camera Control

This game's design relies heavily on different camera modes and with that different movement styles are applied. In designing the prototype three cameras were placed in different zones on the map and numbered 1, 2 and 3. The keys 1, 2 and 3 on the keyboard are then linked to the cameras to trigger the relative camera when the

number is pressed on the keyboard. When the respective numbers is activated the camera switches to that camera mode. On loading the game the last added camera would assume the main camera of the hierarchy and automatically load on startup. Once these tests have passed and provided enough information to prove that the concept works the Kinect script was implemented. A simple prototype was created that tracked the player's head movement and pass this movement to the camera.

5.4 Character Control

As well as triggering a different camera angle and mode the player's character movement needed to change with the camera. For this to work separate movement scripts were created for each camera and once the method to change a camera is called the movement script is also called. The capsule in the diagrams represent the player.

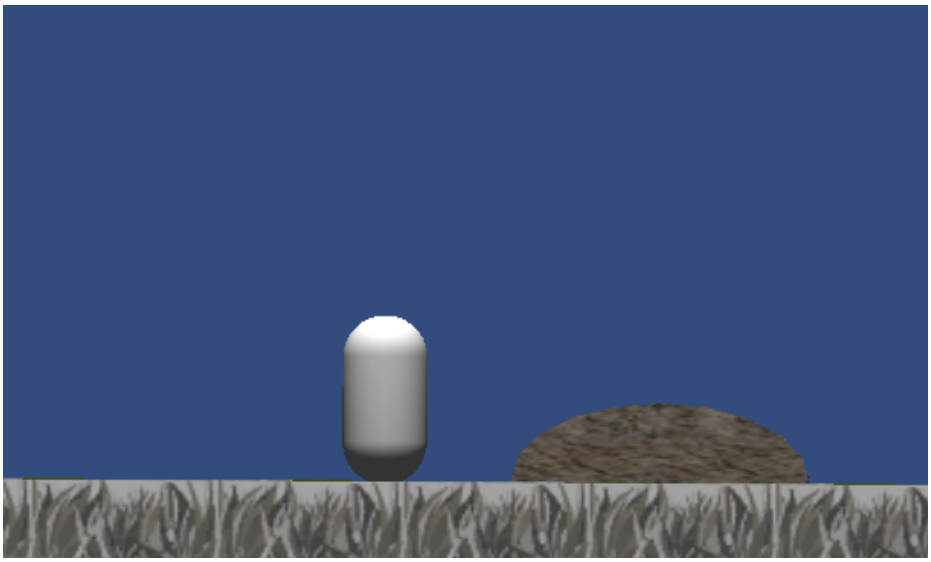

Figure 5.1 – This is taken from a prototype designed to demonstrate the 2D Camera.

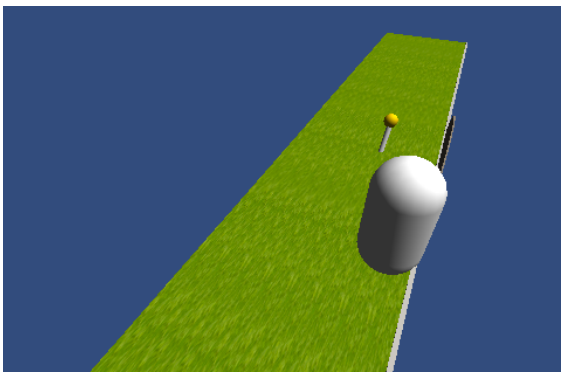

Figure 5.2 – This is the same position with the 3D camera enabled.

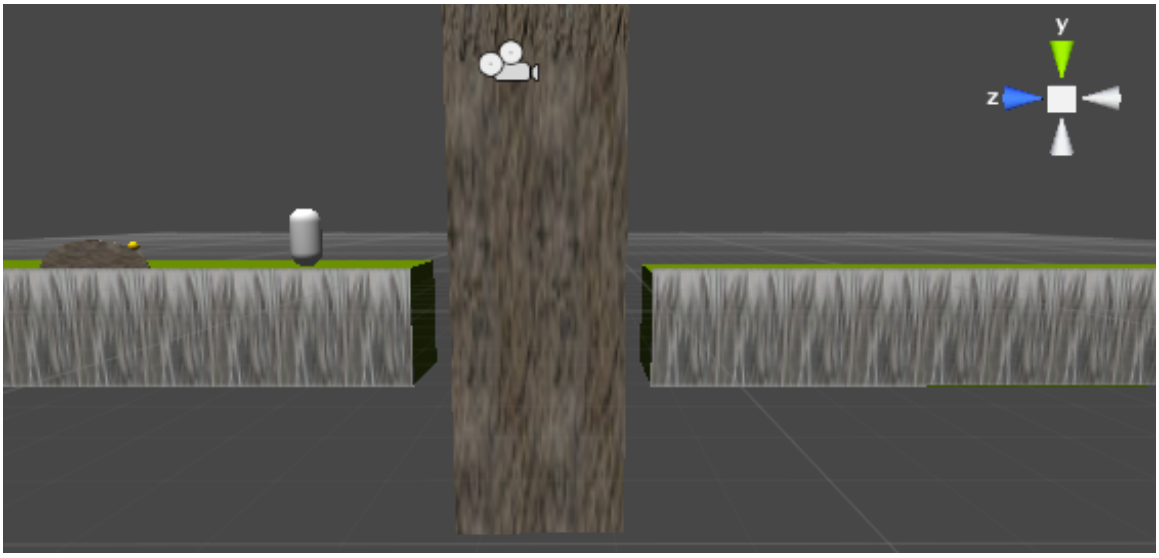

Figure 5.3 – Screenshot of the Kinect mode.

DEVELOPMENT

6.1 Agile Development

While creating the Game Development Document the style of testing was taken into consideration and even though many methodologies of development area are available, Agile seemed to be the most suitable for this project.

"William and Cockburn (2003) state that agile software development "is about feedback and change", and they emphasize that software development is an empirical or nonlinear process, where short feedback-loops are necessary to achieve a desirable, predictable outcome." (Dingsoyr & Dyba 2010, p.2)

Agile Philosophy

- Individuals and interactions **over** processes and tools
- Working software **over** comprehensive documentation
- Customer collaboration **over** contract negotiation
- Responding to change **over** following a plan

(Cunningham 2001)

The Agile development is considered to be a philosophy and principles but no practices are fully defined and set. The philosophy above is taken that what is on the right should be taken into consideration but what is on the left will be valued higher. This philosophy defines outlines an ideal development for a solo developer. Some of the agile methods available are Scrum, eXtreme Programming (XP), Feature Driven Development (FDD), and Crystal. Out of these the eXtreme Programming (XP) is the most attractive method in development. In eXtreme Programming in Action (Roock & Wiley (2002), p. 3) XP is based on "four values: simplicity, feedback, communication and courage".

- Simplicity: Find the simplest solution as this saves on costs and time.
- Feedback: The quality is always kept in control by using Unit Testing and User Feedback.
- Communication: Any problems can be ironed out by communicating and solving issues. (This project has been somewhat a solo project but any problems have been discussed with the project supervisor for guidance and solutions)
- Courage: This is a combination of the above headings that reinforces the ideology of XP.

With these values in mind a “rough” project timeline was played out by referring to the design document and planning how long each section should take. As the project progressed the time allocation would change, as some areas became more time consuming. However the increasing knowledge of the developer allowed for more precise time allocation and projection of development time. Also the encouragement of refactoring code within XP became useful when unforeseen issues arose while the development progressed.

6.2 Test Drive Development

Test-Driven development (TDD) became a major focus for the development and resulting in increasing the development time once it became familiar to the developer. TDD is a process of writing code piece-by-piece and testing each time. With the code failing at first and then writing the correction, sometimes referred to as Unit Testing. The possibility of using Unit testing was researched but only one tool (SharpUnity 2013) is available for this type of development. During the early stages of development SharpUnity was used but it proved to be buggy and unreliable. This was then scrapped but development still followed some of the principals of Unit testing.

This style of test driven development is useful if the code becomes complex and the developer needs to revert to the previous working code. Another feature of this type of testing is that the developer can get feedback on the code instantly. This also adds to simplifying the de-bugging. Considering that the project is a game the testing involved checking the debug console or running the game and testing the code manually. This did prove to be effective in correcting bugs but some didn’t show until play testing was introduced. As shown in *Fig 5.1* “Design / Testing / Development” is continued throughout the projects as to conform to the Agile philosophy.

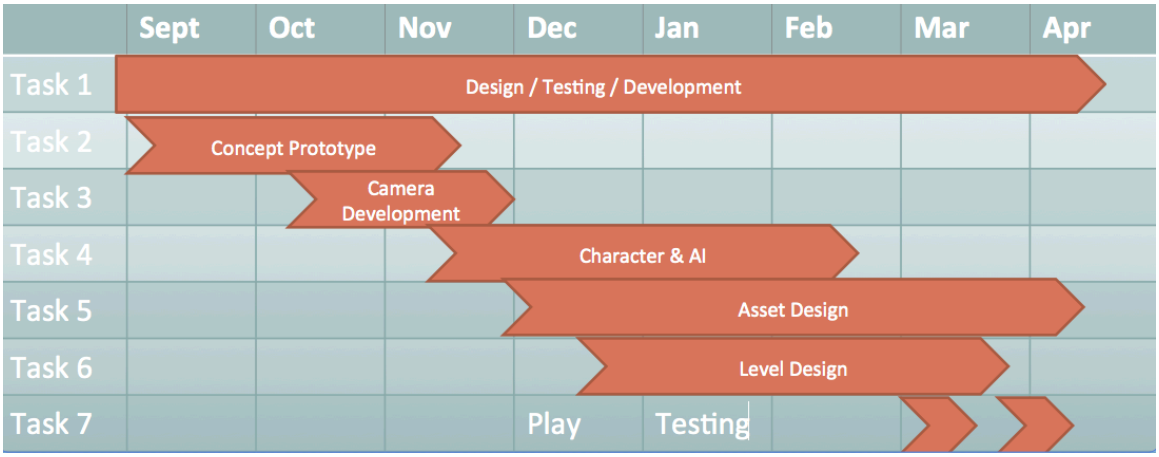

Figure 6.1 – Development Timeline

6.3 Camera Controls

6.3.1 Camera Listener

This is a script that controls the user input and switches the camera. Once a camera is activated, a movement type call is then added to the camera selection script. This enables a different movement style for each camera selection. For the majority of the game the camera is manually switched by the player but at certain points a trigger activates the camera to give the player a different view as seen in the boss battle. Each camera is an object in the game world that can be manually moved and rotated in the **scene** view or like in this case it is programmatically moved.

6.3.2 2D Camera

This camera is always looking at the **scene** from one angle while following the player. The camera uses an orthographic projection mode to give the 2D effect. “With Orthographic Projection the object won't scale by the distance of the camera. So in our **scene**, we will see only one side of the object that faces the camera.” (Wittayabundit 2011, p.19) The camera then tracks a child marker object (CameraTarget2D) of the player. Tracking a marker instead of the actual player object allows for customization and control of the camera. This still tracks the character’s movement, as it is a child of the player object. For instance if you need to raise the camera you just need to adjust the marker and not alter the players position. The use of the target is also useful when the players come to a static area in the game where the camera does not need to move.

A new target is added to the **scene** and once a trigger area is activated the target becomes that static object and does not track the player until leaving the trigger area.

The code for the 2D camera allows for adjustable zoom limits that zooms the camera in and out depending on movement. This effect is achieved by adjusting the Orthographic Projection size to widen the view and give the effect of zooming out. Public variables that define the zoom limits and rate of zoom speed are assigned to the script. If the controller is not moving (horizontal and vertical axis is 0) the camera zooms to the "max2DZoomIn" at the predefined zoom rate until the camera hits the "maxZoomIn". The same applies for the opposite zooming out; if the player is moving then the camera will zoom out until a limit is hit at a predefined rate. The advantage of using public variables is to allow adjustment of the variables while in game play and to get a live view of the camera. The 2D camera also has a battle mode that zooms the camera out to a predefined rate separate from the rates mentioned above and disables the zoom in function. This is used at the end sequence as the player will need to see the enemy at all times as where the traditional camera would not be suitable. Fig 4.3 is an example of the 2D view.

6.3.2 3D Camera

Unlike the 2D camera this 3D camera uses a perspective view that gives a normal 3D view from the camera angle. The camera is assigned a target that is a child of the player, this insures that when the player moves so does the target similar to the 2D target. The angle and rotation of the camera area are assigned within the **scene** view manually. The initial location is then assigned by using the target location. Modifiers are then applied to the location of the target, these modifiers are public variables that are added or subtracted. In this case the camera is set behind the character so the variables are subtracted from the target's x, y, z coordinates. Since these variable modifiers are public they can be changed during gameplay by the designer to get the required camera angle. When designing the camera a zoom function was also needed to create a traditional camera style that is used in most 3D cameras. It also incorporates a lift to the camera to expand the viewing distance as the player moves to add to the character's movement effect. As well as adjusting the x, y and z the speed is also adjusted by applying a Zoom Rate. This is a variable that is added to or subtracted from a certain axis per frame call. A cap is put on the max zoom in and out variables to prevent zooming off the screen or in too far. When the player moves the camera zooms out until the max is reached. When the player is any way zoomed out and stops movement the camera zooms in at a rate until the minimum is reached.

While developing the 3D camera effect a bug was found that caused the camera to cut into a slope as the player walked up a steep angle. This made the game sometime unplayable, as the character couldn't be seen. To resolve this bug a check for the ground type during movement to call a different camera effect. In the "Camera3D1" class the ground is checked and if it is slope ground the camera would move only on the x and z. Tracking the player's movement on the Y axis is enough to not add any modifiers while on a sloped surface.

6.4 Kinect Implementation

Unity 3D does not support the Microsoft Kinect by default; it requires the Kinect SDK available from Microsoft (Microsoft 2013) and the Kinect Unity 3D (Unity Kinect Wrapper 2012) to be installed. The instructions found at Kinect Wrapper indicate that installation of the SDK is essential before running the wrapper. The Kinect wrapper is a set of C# files that call on the Kinect once the Kinect_prefab and KinectAvatar is added to the game's Hierarchy.

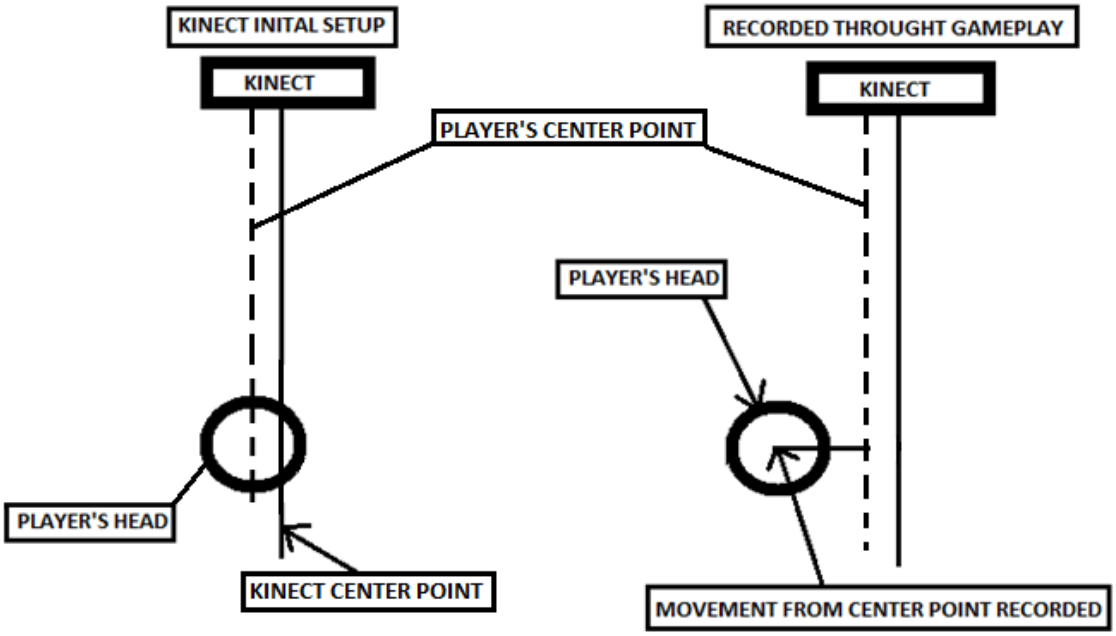

Figure 6.2 – Example of the player calibration process for finding the center point of the play area.

Each time the game starts the player needs to calibrate their body with the Kinect by entering a T pose *Fig 6.2*. Once the Kinect is connected and the skeleton tracking is enabled the KinectAvatar displays each of the player's 19 points on screen. For this project the head is the only point that is needed but any of the other points can be

tracked and used within the game. The HeadMonitor class sets the X, Y and Z variables as a default location when the user configures the Kinect for the first time by using the T pose. These variables are considered the center point of the player's playing area instead of the Kinect's center point. Using this method takes human error into consideration, as the player only needs to configure the Kinect in their perception of the center of the screen See *Fig 6.1*. Once the player moves from this position the movement is subtracted or added to the original center point to get how much the players head has moved. These movement variables alone could slightly move the camera in the game but another layer is applied to these movement variables to amplify the camera movement on screen. Each frame the movement variables are adjusted by public variables in the KinectCameraController class and then applied to the camera within the game. Using this method allows the camera movement to be customized by any game designer using these packages. It also allows for adjustment during gameplay so that the designer can know exactly what the variable can do. Also during testing it became apparent that rotation would be needed as it was noted that as the player moves and plays the game their field of view is always concentrated on the center of the screen. Initially the game camera just moved on the X, Y and Z-axis as the player moved but this only gave a panning effect to the game and it didn't "feel right". In KinectCameraController the rotationAdjust Public variable is set and as the player moves from the center point on the X-axis the rotation adjuster is applied to the camera. This has been quite effective in adding to the game's immersion and also opening new possibilities for a designer.

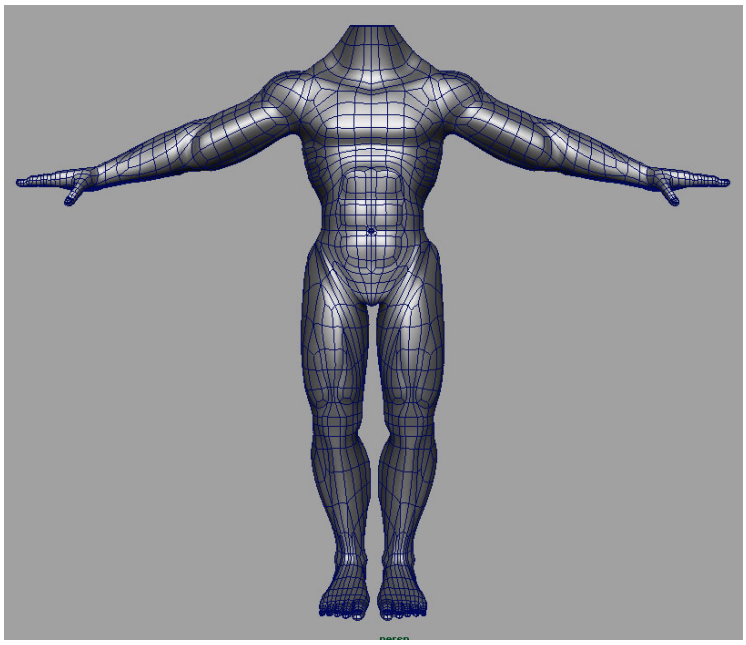

Figure 6.3 – Player T Pose position. (Mutante28 2013)

6.5 Character Movement

6.5.1 Overview

The design for the character movement evolved through the game's development due to bugs found by using certain movement styles. Also the increase of the developer's knowledge with Unity 3D allowed for improvement in character movement. Each movement style is called from the cameraListener, as the player changes the camera the cameraListener calls on the movement assigned to that specific camera. This method of calling movement depending on the camera allows for easy adjustment to the player movement by calling a different style within the script.

6.5.2 2D Movement

The 2D movement script is split depending on what the character is currently doing or the input of the player. The split is mainly sectioned off if the player is in message mode, on the ground, in the air or pushing an object. Each split calls on an animation effect depending on the characters current interaction. The movement is then saved to a Vector3 (Unity 3D position data structure) and the movement is applied to the CharacterController at the end of the method to save on computation. Before detecting input the script checks if the player has a message on screen, if so then all movement input is ignored.

The ground check is used to confirm that the player is on the ground and is then allowed to jump or walk. After some gameplay testing the ability to control the character while in the air was added, without this option it was hard to place the character in specific areas. Instead of using the in game physics an in game gravitation pull is implemented, this is done by decrementing the character's Y-axis until he is on the ground.

A re-spawn method is also added to move the character from one area to another. This is useful when the player falls off the screen in to a hole or runs out of lives.

6.5.3 3D Movement

The 3D movement design went through much iteration as different methods of controlling the character created bugs that were only found during play testing. For instance when enabling character movement and using the in game physics to handle

when the player is in the air it worked while the character was moving. However, when the player interacted with an object such as a box the player would spin out of control. This problem was due to Unity's Rigidbody effect that gives an object realistic physics. To debug this issue a gravitational pull similar to the 2D movement method was applied. Also the box's Rigidbody properties were locked to only move on the X-axis.

6.5.4 3D Kinect Movement

This is similar to the 3D1Movement script that changes the axis depending on the player's input. Since both scripts inherit their movement properties from the CharController3D editing and adding movement scripts is a relatively simple process.

AI CODE

7.1 Overview

Unity's game engine has some built in function to aid in the development of the in game AI. One such feature is the method of searching the game for a game objects tag and returning its location. With this a path can be defined to follow that Tag.

7.2 Spear

The spear is an object that is thrown by the enemy character and if it hits the player three times the player re-spawns and the game **scene** is reset. The spear in the game looks as if it being thrown by the character and the characters AI controls the spear. But in fact all the code for detecting the player and moving towards the player is controlled by AI assigned to the spear.

Unity's engine has certain built in functions that help with creating an effective AI. One that proved to be essential in developing an AI is the FindGameObjectWithTag() call used in the TrackPlayer script. It searches the game **scene** for an object with the defined tag, once found the position is stored by getting the transform of the object. After getting the player's location in the gaming area the spear is then fed these coordinates as the target location during its projectile. The projectile then moves towards the location slightly through every frame while applying the Quaternion.Slerp rotation. SLERP "Spherically interpolate between two vectors" (Unity API 2013). It gets new data points between two sets of data points and results in the object being thrown to dip as it moves in the air. Within the Spear's code a check is set if it has come in contact with the player or if it has reached the target location without any collision. If either of these conditions is met the projectile then drops on the Y-axis dramatically.

While testing the boss battle a problem with the sear not hitting the player in 2D if the character moves towards the projectile. The problem was a mix of the orthographic camera used and the player location in 2D. If the spear follows the path to reach the player, in the 3D view this would be acceptable but for the 2D view it does not make sense. Instead of following along the axis in 3D a check if the camera is currently in 2D mode. If so the starting Z location of the spear is set to the players Z location. This proved to be effective, as the player could not see the axis changing and yet provided the expected 2D playing experience.

7.3 Follower

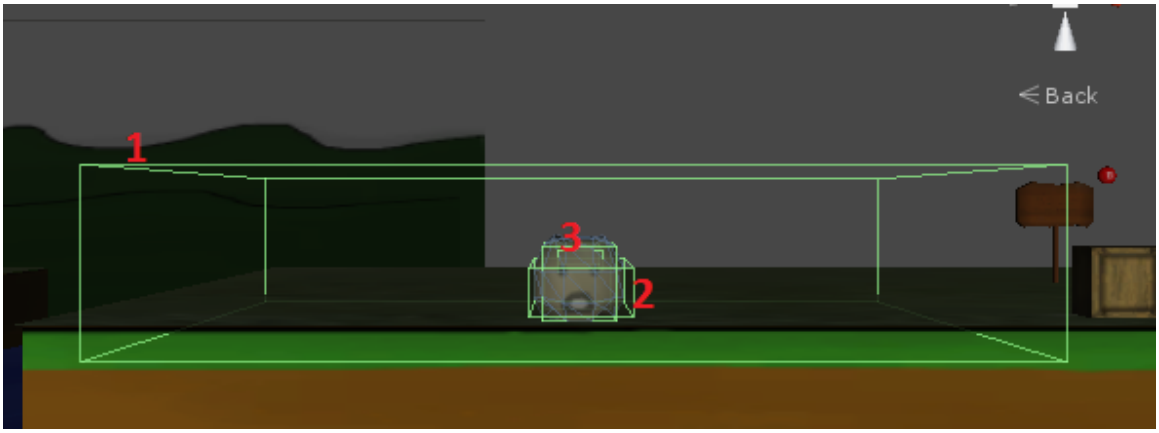

Figure 7.1 – Follower Game Object

This is an enemy object that once the player comes in contact with its field of view the object moves towards the player. If a collision is detected then the player loses a life, the player can also jump on the objects head and destroy the object. The AI for this inherits the TrackPlayer class to find the players location on the map. The object is made up of three main properties. When the player enters the trigger zone see **1** in fig 8.1 the object stores the players location per frame cycle while it is in this zone.

Number 2 in fig 8.1 is a collision box that decrements the players lives once it comes into contact with the player. A timer of 3 seconds is also in place to prevent continuous decrementing while the player is in contact. Without this in place a life would be decremented for each frame while the player is in contact.

Number 3 in fig 8.1 is another collision box that allows the player to destroy the object by jumping on it. The player uses a ground detector to return what type of ground the player is currently standing on. Once 3 collides with the ground detector the follower object along with its children are destroyed and a particle effect is played.

Asset Code

8.1 Overview

Asset code is a section of code that is assigned to interactive objects in the game. These are sectioned off from other scripts as they are activated by the player and do not contain any intelligence.

8.2 Moving Box

Adding the ability to move a box was essential for game progression and also adds to puzzle complexity. The player could walk towards a box and move it by moving in front of the box while holding a button. This seemed simple on paper and the development time slot set aside for the box movement was short. However after developing the first script using Unity's rigidbody physics problems started to form during game play. If the player approached a box and held the push button while moving in the box direction the physical contact between the player and the box was too powerful. This ended up in throwing the box off the screen with too much power. Lower speed variables were added but it still didn't resolve this issue.

Instead of using the physics the developer removed all physical elements from the box game object and instead allowed the box to become a child of the player. This solution allowed for solid movement and the box speed depended only on the player's input. At first this posed as the best solution but during testing the player would be able to push the box but both the box and player would end up randomly rotating. Once the box is a child of the player the box would act as leverage in the player's physics. Again methods to try and control the box and player were added but it still didn't resolve the issue.

The next solution was to allow the box to have limited physics. While the player was in contact with the box and the push button held the player's movement was also applied to the box. This method took quite a bit of development resources but the end result was just about correct until the Kinect was enabled.

8.3 Kinect Issue

The development process was broken down into areas that needed to be complete before moving on to the next. The first on the list was the Microsoft Kinect development process; once this was written and tested the Kinect was disconnected from the computer until the final few phases. Considering that the Kinect was disconnected some sections of the code were tested without the Kinect present. Once the Kinect was connected major bugs were found that rendered the game unplayable in some areas. Unity's function Collider.OnCollisionStay calls a method when a collision with another object is detected. This was used extensively as checkers in the game for the box movement to any trigger zone (Sing, Boss Battle etc.). The box movement relied heavily on this as it could return that the player was in contact with the box and the box could be pushed. Without the Kinect the check for trigger contact was called 60 – 70 times a second but once the Kinect was enabled this dropped to 4 – 5 times a second. The code that worked perfectly became incompatible with this type of return time and ended up with choppy response in game or in some cases no response. A refactor of the code was needed and instead of using the Collider.OnCollisionStay a Boolean is set to true if the player entered the collider (Collider.OnCollisionEnter) and a set to false if the players exits the zone (Collider.OnCollisionExit). On the game's Update call the Boolean would indicate if the player is in the area or not. This solution proved invaluable to the game as without this the game would have not been completed. This solution is a prime example of where the Unity API shows its qualities as an easy to understand and search.

8.4 Rock Area Bug

One of the first puzzles designed for this game was the Rock Puzzle. It required the player to walk up a hill as a rock is rolling down. The player is required to switch into 3D mode to walk around this 2D rock. The Rock is given a rigidbody physical attribute and is triggered when the player passes a certain point. The developer found a bug that once the object is triggered to move the object would go out of shape *Fig 8.1*. Originally a bug was considered to be a problem with the developers code, however after testing the engine environment it became clear that the problem was with an object being created off the view of the game area. This was found by running the game with the object in the **Scene** view and then without. To avoid this bug the in game camera would shift into 3D mode for 1 second to show the object falling. As well as removing the bug it also tells the player that a rock is coming down the hill.

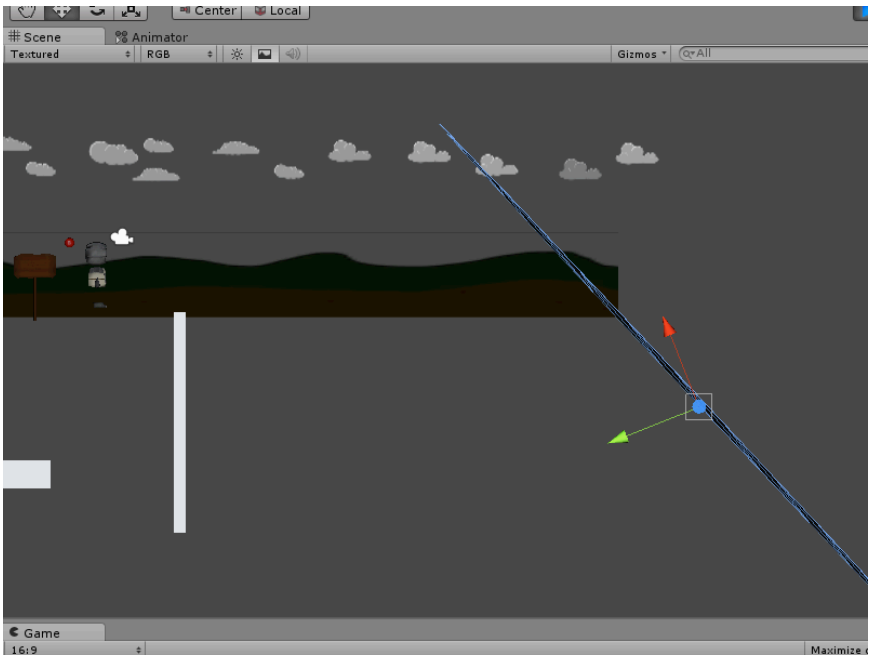

Figure 8.1 – Bug that stretched the rock instead of creating a small round cylinder.

8.5 Parallax

Parallax is a method of layering images over each other and moving each layer to give the illusion of 3D. It is a technique used in classic games from the 90s but recently it has been popular in the mobile game market. Unity does not have a parallax mode included in the engine but it is for sale on the Unity Asset Store (Unity Store Parallax 2013). Considering that the only available option is to buy the effect the author decided to implement a version himself. Originally the parallax was going to be implemented by adding rectangle layers to the background and moving these with the player as well as repeating the layer as they moved off the screen. The original code that repeated the layer only worked if the player is moving forward, once the player moved backwards the rectangle would move off the screen. Instead of moving the rectangle background the texture on it would only move (This is the process of moving the image on the rectangle). The code monitors the player's location and if player movement is detected that movement is then applied to the texture. If the texture movement needs to move faster the movement is multiplied by a defined variable before applying the texture movement. This is a successful method of getting the effect and will help others in using a free version of the parallax in Unity.

GAME ASSETS

9.1 Modeling Resources and Software

Since the increase in indie game development allot of websites provide readymade and animated models to import into a game. The problem with this is that these models usually range from $50 up and it can also limit your games characters movement. For instance models from ProtoPack (FroGames 2013) that provided simple models at a reasonable price, but the problem is that the models don't supply an animation for pushing an object. Any other sites that provided models that included a push animation were too expensive or the model was not suitable. An alternative to getting pre made models would be to build them using easy to understand software, one of which is 3D Coat (3DCoat 2013). This software allows the user to sculpt a model from shapes and easily import into Unity. The problem with this software is that it wasn't compatible with a 64Bit version of Windows 7. Next solution for model creation would be to use one of the popular modeling software packages that are used in the gaming industry. After some research two software packages seemed to dominate the gaming modeling industry 3ds Max (Autodesk 2013) and Blender (Blender 1995). Both packages allow you to create and animate models with an abundance of tutorials and books available. However 3ds Max is a commercial product and cost over $200 while Blender is open source and free.

9.2 Brief History of Blender

A Dutch game animation company NeoGeo created an in house 3D design application that would become Blender. After a few years the company started working on free software that could create interactive 3D content and provide a commercial version for publishing and distribution. After bad sales and unforeseen issues the investors pulled the project but the creator wanted to keep the project going and created the Blender Foundation. After some arrangements the software was released under the GNU terms

on October 2002 and made available to everyone. The project has gone through multiple updates and added many features such that make this software an essential tool in independent game development. (Blender 2011)

9.3 Using Blender

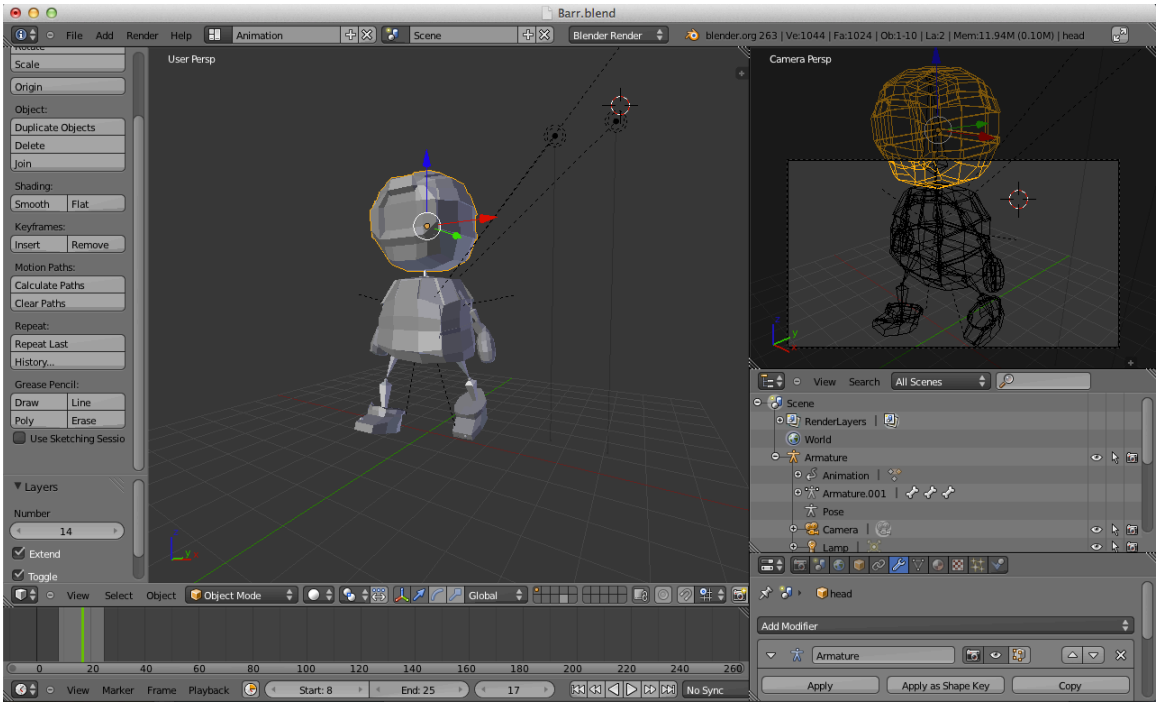

Figure 9.2 – Blender Modeling Software

Creating a model in Blender using box modeling technique from “Blender Foundations” (Hess 2010) proved to be an effective and rapid modeling technique. Blender’s layout at the start can be confusing due to multiple windows that are presented on opening the application. Blender allows for multiple windows to be opened in the Blender’s layout as modeling may require more than one tool to be used at the same time. Each window in Blender has a Header that allows access to the controls and menus within that tool. To access Blender’s tools the use of hotkeys is essential and learning them can cut down the production time of a model. To create a simple model a mesh needs to be added, this is a 3d object that you can shape into the model. Box modeling is a process that “You can add geometry to it through subdivision, loop cuts, and extrusion, but you will never delete a face, edge, or vertex in a way that leaves a hole.” (Hess 2012, p.85) Once

you have a finished product you can apply the modifier “Subdivision Surface”, this is a process of taking the mesh and dividing its triangles and quadrangles into smaller tessellations and smoothing the mesh.

9.4 Blender Texturing

Blender’s built in texturing tool allows the user to paint the model once the seams are marked on the model. Marking the seams is the process of breaking the model’s mesh into smaller areas and producing a UV Map of the model. The map created is a flattened out version of the model that can be easily painted and processed. On exporting the model the texture is automatically applied.

TESTING

10.1 Play Testing

Play testing involves getting people to play the game before the official release to find bugs or problems in the game. In this case the author asked people he knew to try out the game and give feedback. While the game is being played the author took notes on how the player played and tried to not give the player any information on how to play. This simulated the process of actually releasing the game and how people may react. These tests were carried out as different milestones were met to improve the game throughout development and prevent repeating any problems a player may have. The testing was not started until mid March even though some gameplay was possible before this date. The author felt that without user-friendly graphics a player would judge the game more on looks than actual gameplay. Before the player starts a quick note on the players experience with the Kinect and games played is taken. After finishing the test a playtest document is then given to the player to fill out. Here is an example of one of the documents filled out by player 1 Fig 10.1. All documents have been compiled into the tables below also including the test administrator's notes during gameplay. During this section the author is referred to as the test administrator.

Playtest Document

Date: 10 March 2013

Sex: (M) / F

Age: 25

Gaming Experience: Newbie ✓ Casual Gamer Hardcore Gamer

Likes: I like the concept of the game of how the character moves and the Kinect control

Dislikes: Hard Puzzles

3D control needs to be limited
the camera buttons don't feel right

Suggestions: Limit on 3D view

Button layout

Rating From 1 - 10: 7

Figure 10.1 – Example of the Playtest Document. (Player 1)

10.2 Player Test 1

This section of testing is the first of two. It demos three puzzles that use the main functionality of the game mechanics. Considering that this game is aimed at children and adults a mix of player's ages and sex has been compiled.

Test Number	Player 1
Date	10/03/2013
Sex	Male
Age	25
Gamer Experience	Casual Gamer
Game Rating	7/10

Player Description: This player has never used the Kinect and is interested in how it works.

Gameplay Notes: On startup the player read the instructions and followed the level. He tried to use 3D as much as possible and didn't understand the limitations of each view. When he enabled the Kinect the head tracking had trouble finding him, this was due to the player being tall. This section level was completed above average time as some extra time was spent trying to solve the more difficult puzzles.

Likes: "The concept is a good idea" and "looking forward to the end product."

Dislikes: "Hard puzzles", "3D control need to be limited", "camera buttons don't feel right"

Player Suggestions: "Button Layout"

Rating 7 out of 10

Test Number	Player 2
Date	10/03/2013
Sex	Female
Age	25
Gamer Experience	Newbie
Game Rating	6/10

Player Description: This player has played some exercise games on the Kinect but not many games that use a controller.

Gameplay Notes: When the game started she read the instructions and instantly used the Kinect without knowing when it should be used. She started to move her arms as the games that she has previously played on the Kinect use full body tracking. The test administrator informed the player that the game only uses head movement and that it is needed only in certain areas of the game. She then enabled the 2D camera and started to play and read the gameplay information. She got to the area where she needed to jump and switch into 2D mode to land on a rock but found it “impossible”. We ended here as it was close to the demo.

Likes: “The game looks pretty” and “the body movement”

Dislikes: “Hard puzzle” and better description for the Kinect control better

Player suggestion: Better description of Kinect movement. Make the 2D jump puzzle easier.

Rating 6 out of 10

Test Number	Player 3
Date	10/03/2013
Sex	Male
Age	14
Gamer Experience	Hardcore Gamer
Game Rating	4/10

Player Description: This young player would be a gamer that spends a lot of time playing games online and offline. He has also played 2D games on the Nintendo DS but he has never used the Kinect.

Gameplay Notes: The player quickly glanced at the instructions on how to play and tried to switch views constantly. Before the test he was informed that the test administrator couldn't give any information he took his time and tried to figure out the game. After a few minutes he realized that the first puzzle needed 3D enabled to solve it. He then tried to push a box in the 2D camera mode but the character started to spin out of control. The game was reloaded at this point and the bug didn't reoccur. He ran through the Kinect process easily as he read the game information this time. But he did take extra time trying to solve the final puzzle. After finishing the demo the player was a little frustrated as he didn't read the instructions at the start and this may have affected the rating.

Likes: The head tracking was his favorite.

Dislikes: “Hard to know what to do in the game”

Player suggestion: “Make the game easier”

Rating 4 out of 10

10.2.1 Analysis & Changes

These tests provided enough information for adding improvements to the gameplay and also the button layout. Considering that the code has been designed in a way that changes are easily applied these suggestions did not add to development time.

10.3 Player Test 2

This section is a test on the same section as the last with additional puzzles. The three players from the last test have been asked to play again and give their opinions on the game. Also an additional two people have been asked to play as the test still needs to express the opinions of someone that is playing the game for the first time.

Test Number	Player 1
Date	01/04/2013
Sex	Male
Age	25
Gamer Experience	Casual Gamer
Game Rating	8/10

Player Description: Player has played the previous version of the game.

Gameplay Notes: The player played through the game still taking time at some puzzles but yet solving them. He used the Kinect where needed and sometimes switched to Kinect view while the 3D camera view recharged.

Likes: “Kinect effect” “Game Controls” “General Camera Improvements”

Dislikes: “Some Puzzles Take Too Long”

Player Suggestions: Simplify the Rock puzzle

Rating 8 out of 10

Test Number	Player 2
Date	01/04/2013
Sex	Female
Age	25
Gamer Experience	Newbie
Game Rating	8/10

Player Description: Player has played the previous version of the game.

Gameplay Notes: This time the player knew that the Kinect didn't track the players hands and she played through the first three puzzles that she had already seen in the previous test. The rock puzzle took her only two attempts to figure it out and she finished the game in average time.

Likes: “Head tracking looks good” “Smart puzzles”

Dislikes: “The controller buttons are had to remember”

Player suggestion: “Longer level and more puzzles”

Rating 8 out of 10

Test Number	Player 3
Date	01/04/2013
Sex	Male
Age	14
Gamer Experience	Hardcore Gamer
Game Rating	6/10

Player Description: Player has played the previous version of the game.

Gameplay Notes: The player again passed the gameplay information instantly and started to play. He found the Kinect puzzle at the start had to get through but he solved the puzzles after that in average time. The first part of the rock puzzle took him only one try and the second part stopped him. He passed the last section in average time and knew what to do instantly.

Likes: "The game looks good" "The controls work"

Dislikes: Kinect puzzle is hard

Player suggestion: Suggested a "Longer game"

Rating 6 out of 10

Test Number	Player 4
Date	01/04/2013
Sex	Male
Age	19
Gamer Experience	Hardcore Gamer
Game Rating	7/10

Player Description: This player plays a lot of games and is a big follower of independent games.

Gameplay Notes: The player read the gameplay information and started the game. He was a little confused with the controls at the start but soon became familiar with them. The Kinect puzzle was solved in average time and he commented that he liked the

"Kinect idea". He solved the remaining puzzles in average time. He did however have an issue with jumping from one box to another.

Likes: "Kinect idea works well"

Dislikes: "Character jumping isn't working"

Player suggestion: Jumping not working

Rating 7 out of 10

Test Number	Player 5
Date	01/04/2013
Sex	Male
Age	22
Gamer Experience	Newbie
Game Rating	8/10

Player Description: This player is someone that hasn't played games in a long time.

Gameplay Notes: He took his time reading everything on the screen and found the first puzzle difficult but then realized how the game is designed. The test administrator did need to give him some guidance in the Kinect puzzle as he found it hard to use the controller as well as standing in the correct area. He pushed the boxes without any guidance but the rock puzzle was another one he couldn't manage without some guidance. We finished the demo early but he did enjoy the experience.

Likes: "Being able to move in the game" (Kinect tracking)

Dislikes: A little difficult

Player suggestion: --

Rating 8 out of 10

10.3.1 Analysis & Changes

After this round of play testing the players thought the game is fun and advised on adding to the game level. Some complaints of making the game easier has been noted but advised that it is a puzzle game and it is made to make you think about your actions. The jumping bug that Player 4 found was due to the character not being bale to move while in the air.

10.4 Player Test 3 (Demo Day)

This section is more of an observation on demo day in the university. General feedback was positive with considerable interest in the game's original concept and application. Some people showed interest in downloading the game and extending the mechanics.

CONCLUSION

11.1 Conclusion Overview

The aim and primary motivation behind this project was to use head tracking in an original game concept while giving back to the online community. With the game mechanics now available online it will hopefully help others in developing games of their own. It may even help in creating new head tracking mechanics that I never even considered. Uploading videos of this project online has also connected me to a company that work on new innovative projects in the human interaction sector.

The research carried out during the project has given a real insight of how the original VR Desktop was created. Also how people reverse engineered consumer electronics to create something new.

The successful implementation of this project has provided me with invaluable skills and knowledge in various roles that will help with in future projects. I now fell that I have enough knowledge to take on and manage new projects in Unity development considering that I have a greater knowledge in its limitations. This project has also allowed me to be proficient in writing C# code and designing scripts for games. As well as the ability to code, the ability to create 3D models is something that will prove to be useful in the future.

Using the Kinect has made me think about new areas that the Kinect could be used in. Such as the medical sector to track human hands and simulate medical procedures.

11.3 Improvements to the Project

The game did meet my expectations and even exceeding in some areas but certain areas need some improvement:

11.3.1 3D Control Redesign

The current setup is working but some improvements are needed in controlling the character while in the air. Attempts have been made to tweak the control but due to time restraints these tweaks didn't take effect. I would recommend at redesigning the gravity in the 3D movement as some bugs are due to the constant Y-axis pull. Setting aside a different gravitational pull when the player is in Jump mode may solve this issue.

11.3.2 3D Camera Tweaking

The camera is one area that took a lot of the development resources but even though the outcome is effective the 3D camera can always be tweaked. Certain bugs for instance when the player re-spawns the camera can sometimes jump. This is an inconsistent bug but it does exist. I would advise providing a check when the player is in re-spawn mode and controlling the camera in a different way.

11.3 Future Work

Even though all the goals were completed, this project still has more potential for changes. The use of new upcoming innovate technology could be used in the game. Such as the LEAP Motion Controller (Leap 2013) that has the ability to fully track the users hands to never seen before precision. Also the Oculus Rift (Oculus 2013) may be used as a VR headset. Considering that time played a major factor in limiting certain aspects here are a list of recommendations that could be accomplished in the future.

- Moving platforms
- Greater Intelligence AI (Path Finding)
- Additional Enemy
- Add New Hardware (Leap, Oculus Rift)
- Add Audio
- Use the current head tracking tools to create different views

REFERENCES

1. Nintendo (2010) 'Looksley's Line Up', [online], available: http://www.nintendo.com/games/detail/xFv-4ah7Igv1-HewgD8wOyWAyRVbs3EM [accessed 9th Sept 2012].
2. Sony (2008) 'Echochrome', [online], available: http://us.playstation.com/games/echochrome-ps3.html [accessed 9th Sept 2012].
3. Lee, J.C.; , "Hacking the Nintendo Wii Remote," *Pervasive Computing, IEEE* , vol.7, no.3, pp.39-45, July-Sept. 2008 doi: 10.1109/MPRV.2008.53
4. Nintendo (1996) 'Mario 64', [online], available: http://www.nintendo.com/games/detail/48czfMMYIp_0_alQYMIOS7sgrvUdDq1O [accessed 9th Sept 2012].
5. Gimp (1996) 'GIMP', [online], available: http://www.gimp.org/ [accessed 20th Oct 2012].
6. Blender (1995) 'Blender', [online], available: http://www.blender.org/ [accessed 9th Sept 2012].
7. Microsoft (2010) 'Kinect', [online], available: http://www.microsoft.com/en-us/kinectforwindows/ [accessed 4th Sept 2012].
8. Unity 3D (2005) 'Unity Homepage', [online], available: http://www.unity3d.com/ [accessed 5th July 2012].
9. Alan Holmes (2011) 'The Development of a Cost Effective Tracking System for Use Within a Networked Multi-User Three Dimensional Audio-Visual Environment' - MSc. In Music Technology
10. FaceApi (2012) 'FaceAPI', [online], available: http://www.seeingmachines.com/product/faceapi/ [accessed 13th Aug 2012].
11. Zhengyou Zhang; (2012), "Microsoft Kinect Sensor and Its Effect," MultiMedia, IEEE , vol.19, no.2, pp.4-10, Feb. 2012 doi: 10.1109/MMUL.2012.24
12. Unreal Engine 3 (2012) 'Unreal', [online], available: http://www.unrealengine.com/features/ [accessed 27th July 2012].
13. Unity 3D (2012) 'Unity Store', [online], available: https://store.unity3d.com/ [accessed 29th July 2012].
14. WalkerBoyStuido (2012) 'Walker Boy Online Tutorials', [online], available: http://walkerboystudio.com/html/unity_course_start_here__free_.html [accessed 17th Aug 2012].

15. Unity 3D API (2012) 'Unity 3D API', [online], available: http://docs.unity3d.com/Documentation/ScriptReference/ [accessed 01th Jun 2012].
16. Unity 3D Forum (2012), [online], available: http://forum.unity3d.com/ [accessed 01th Jun 2012].
17. Blackman, Sue. (2011) Beginning 3D Game Development with Unity: All-in-one, multi-platform game development. 1st ed., New York: Apress
18. WalkerBoyStuido (2012) 'WalkerBoyStuido Design Document', [online], available: http://vimeo.com/album/1567704/video/22275477
19. Blender (2011) 'Blender History', [online], available: http://www.blender.org/blenderorg/blender-foundation/history/ [accessed 01th Feb 2013].
20. The Codehaus (2003) 'Boo Language', [online], available: http://boo.codehaus.org/ [accessed 01th Mar 2013].
21. Microsoft (2003) 'C Sharp', [online], available: http://msdn.microsoft.com/en-us/vstudio/hh341490.aspx [accessed 01th Sep 2012].
22. HoBlog (2013) "C# vs UnityScript" http://www.holoville.com/blog/?p=820 [accessed 20th Oct 2012].
23. Microsoft (2013) "Kinect SDK" http://www.microsoft.com/en-us/kinectforwindows/develop/overview.aspx [accessed 20th Jun 2012].
24. Unity Kinect Wrapper (2012) 'Unity Kinect Wrapper', [online], available: http://wiki.etc.cmu.edu/unity3d/index.php/Microsoft_Kinect_-_Microsoft_SDK [accessed 27th July 2012].
25. Dingsoyr & Dyba (2010) Agile Software Development. 1st ed., Springer
26. Cunningham (2001) 'Agile Manifesto', [online], available: http://agilemanifesto.org/iso/en/ [accessed 02th Sep 2012].
27. Lippert Roock & Wolf Wiley (2002) eXtreme Programming in Action – 1st ed., Wiley
28. Wittayabundit (2011) Unity 3 Game Development Hotshot – Packt Publishing
29. Unity Store Parallax(2013) 'Parallax', [online], available: http://u3d.as/content/pint-sized/parallax/1R3 [accessed 03th Mar 2013].
30. FroGames (2013) 'ProtoPack', [online], available:http://www.frogames.net/content-packs/protopack.html [accessed 20th Feb 2013].
31. 3D Coat (2013) '3D Coat', [online], available: http://3d-coat.com/ [accessed 22th Feb 2013].
32. Autodesk (2013) '3ds Max, [online], available: http://www.autodesk.com/products/autodesk-3ds-max/overview [accessed 23th Feb 2013].

33. Ronald Hess (2010) 'Blender Foundations' – 1st ed., Focal Press
34. Mutante28 (2013) 'T Pose Model', [online], available: http://mutante28.deviantart.com/art/3D-Male-Model-quot-T-Pose-quot-150435482 [accessed 10th Apr 2013].
35. Wikipedia (2013), 'Xbox 360 Controller' http://en.wikipedia.org/wiki/Xbox_360_Controller [accessed 10th Apr 2013].
36. SharpUnity (2013), 'SharpUnity' http://wiki.unity3d.com/index.php?title=SharpUnit [accessed 01th Sep 2012].
37. Ocolus (2013), 'Ocolus Rift' http://www.oculusvr.com/ [accessed 10th Feb 2013].
38. Leap (2013), 'Leap Motion Controller' https://www.leapmotion.com/ [accessed 7th Jan 2013].

BIBLIOGRAPHY

1. Blackman, Sue. (2011) Beginning 3D Game Development with Unity: All-in-one, multi-platform game development. 1st ed., New York: Apress
2. Dingsoyr & Dyba (2010) Agile Software Development. 1st ed., Springer
3. Ronald Hess (2010) 'Blender Foundations' – 1st ed., Focal Press
4. Lippert Roock & Wolf Wiley (2002) eXtreme Programming in Action – 1st ed., Wiley
5. Wittayabundit (2011) Unity 3 Game Development Hotshot – Packt Publishing
6. Sharp, John (2005) Microsoft Visual C# 2005 Step by Step – Microsoft Publishing