Jason Dansie

Game Development in Unity

Game Production, Game Mechanics and the Effects of Gaming

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

01 August 2013

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Jason Dansie Game development in Unity 34 pages 1 August 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialization option	Programming
Instructor(s)	Markku Karhu, Head of Degree Programme

The goal of this thesis is to examine how video games are designed and to see how different game mechanics work and how to use them in the development of a game, as well as examine what are both the positive and negative effects games have on adults and children.

This thesis looks at how games in general are developed in Unity, a 3D game engine which has become not only popular but a standard in the gaming industry. The thesis describes how the interface in Unity is used to quickly generate game environments, how scripts are used for logic, game interaction and other game mechanics. It demonstrates how a few of these games were made in Unity and also deployed to several different devices and operating systems with several specific games target for children ages four to eight years of age.

Further the thesis shows how deployment to several mobile devices helped to allow users of all ages and backgrounds to be able to test the games. Giving the programming group insight and ideas for improvements to the games being produced. Finally it also takes a look at game addiction, violence in games and the positive effects of games.

In conclusion knowing how violence in game affects a person. As well as how to use games to create positive affects in people. A game developer will know how the games they produce affect the audience they have created the games for. Also a concerned parent can chose a game they feel their children can learn from or use as a positive outlet for anger.

Keywords	games, Unity, 3D games, game engines

Contents

1 Introduction 1

2 Unity 3D Game Engine 2

2.1 The Scene View 2
2.2 The Game View 3
2.3 The Hierarchy View 3
2.4 The Project View 4
2.5 The Inspector View 5
2.6 The Physic Engine 6

3 Common Game Mechanics 7

3.1 Collision Detection 7
3.2 Finite State Machine 8
3.3 Timers 9
3.4 Path Finding 9
3.4.1 Dijkstra's Algorithm 10
3.4.2 A* Search Algorithm 10
3.4.3 D* Search Algorithm 11

4 Games Developed in Unity 12

4.1 Brick Breaker 12
4.1.1 Brick Breaker Game Mechanics 12
4.1.2 Touch Input 13
4.2 Flips 14
4.3 Mouse in a Maze 15
Mouse Maze Game Mechanics 16
4.4 Flight Game 17
4.5 Candy Monster 18
4.6 Grabs 20

5 Deploying Games to Mobile Devices 21

5.1 Deployment to Android Devices 21
5.2 Deployment to Apple Devices 22
5.3 Deployment as a Cross-platform Game 22

6 Writing Technical Training Material 24

6.1 Graphical User Interface (GUI) 24
6.2 The Scoring System 27
6.3 Main Game Character 27

7 Game Testing Results 29

7.1 A Child's Point of View 29
7.2 An Adults Perspective 29

8 Game Addiction 31

9 Violence in Games 33

10 Positive Effects of Video Games 34

11 Conclusion 35

References 38

1 Introduction

The topic of this thesis is game development with Unity, game production, game mechanics and the effects of gaming. Each topic was covered and discussed in a project proposed by Metropolia University of Applied Sciences (UAS). For this project two artists and four programmers were hired to create video games and a tutorial that Metropolia UAS could use to promote its game programming program. The group was able to decide what the game did and how the tutorial was written. The group chose to make a single game environment where six smaller games could be chosen from and played. The thesis work and report were based off that project.

Unity 3D game engine was used during this project as it has the capabilities the group needed for deploying the game made to as many mobile devices as possible. Unity can deploy the game made to Windows, UNIX, Mac, iOS, Android and Windows 8 phone. As this is the only game engine out currently that can deploy to all these operating systems at once Unity was the choice for this project.

There is a multitude of reasons behind the motive for this research. Firstly the game industry in Finland is becoming quite large due to several popular games released by Finnish companies and this alone would be enough encouragement to learn game development techniques using Unity. The second is Metropolia has a need to demonstrate its courses offered concerning game development, game art and game programming as well as set itself as a leader in the country for training students in this field. Lastly it is an area of my own interest. My first experience with game programming; it was back on an Atari 2600. My friend's mother had a book that had examples of how to do specific game mechanics and she had written a game or two. The introduction of this book to me sprung forth my complete fascination with games and programming, hence my further research into game development.

This thesis begins with explaining Unity game engine and the graphical user interface used to produce the games. The thesis will then go into the mechanics of each game and some of the code that the games use. The programming language used for the games is C#, specifically for scripting in the games. The thesis will also cover deploying the game to multiple platforms and mobile devices after which the thesis will cover violence in games, game addiction and the affects games have on children.

2 Unity 3D Game Engine

Unity 3D is a game engine and complete integrated development environment (IDE) with an integrated editor, asset workflow, scene builder, scripting, networking and more. It also has a vast community and forum where any person wanting to know and learn to use Unity can go and have all their questions answered.

There are five main views used in the Unity editor to get all the work done, the project view, scene view, game view, hierarchy view and inspector view, all of which are explained in more detail below.

2.1 The Scene View

The scene view is one of the most used views as this is where all the game objects are placed and scenes for the game are built.

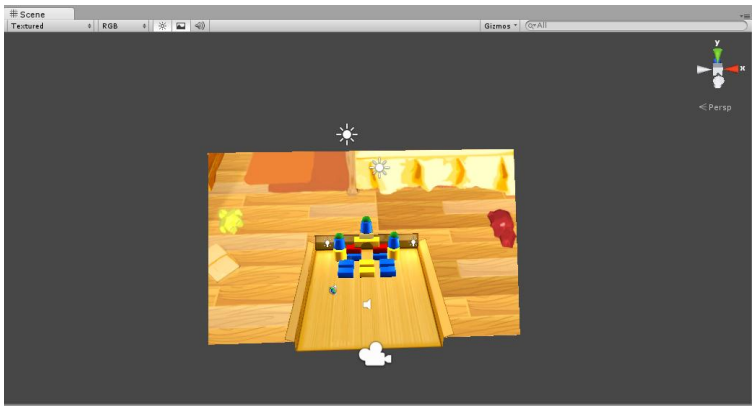

Figure 1: Screenshot of the scene view in Unity with a level from brick breaker. Screenshot [3].

In figure 1 the scene view is shown with a level from the Brick breaker game. The level is designed in 3D, although it is played more as a 2D game. “To create a 2D game in a 3D environment, the one degree of freedom is removed. The y axis is all but ignored and the camera is placed in orthographic mode looking down on the screen as though it were played on the top of a table.” [1] In the project the view was turned to give it a 2D feel which is used for all the games built during this project. The level looks very different in this view as it would in the game.

This view allows the programmer to move through the entire 3D world the game is built in. To help position objects, Unity can snap to specific increments when dragging. “To use snapping, hold down *Command* (Mac) or *Control* (PC) when using the Translate tool (*W*) to move objects in the Scene view.” [2] All areas of the game can be viewed; even ones a

player of the game cannot get to. The 3D world is not restrained in any way if an object was thrown in any direction in this screen it would theoretically go forever. This is really only limited by the computer or device that the game is run on. [3]

2.2 The Game View

The game view is what user will see when the game is started. There are several options for this window. Across the top of the window there are several button/drop down menus which can change things from the perspective, full screen, and gizmos shown in the game view.

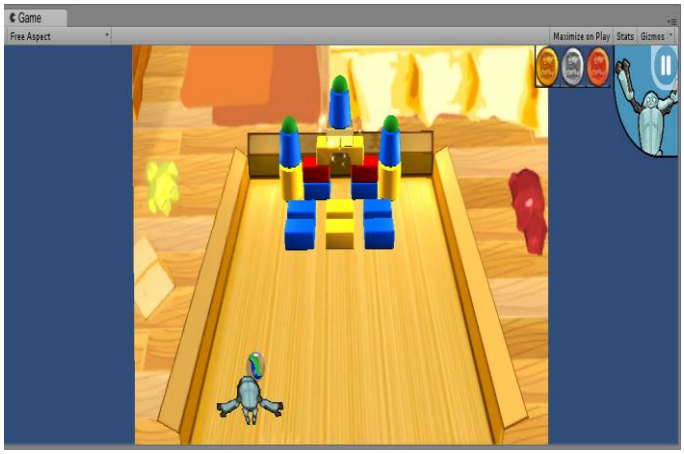

Figure 2: Screenshot of the game view with a level from brick breaker. Screenshot [3].

As seen in figure 2 the character for this game shows up and can launch the ball to destroy the brick castle in the distance. Also the pause menu and score system can be seen. Unlike in the scene view the player cannot see that there is a huge 3D world that contains this little level they play in. When the game is running, game testing happens in this view.

Brick breaker can be ran, paused and restarted to help in the testing process. One other view that is useful when testing the games is the console view (not shown). In that view all the output from the game can be seen and debugging messages can be used to help track down pesky bugs in the code.

2.3 The Hierarchy View

The hierarchy view is where all the objects in the game can be created, accessed, grouped and manipulated to make the game. When the project is saved, the objects are saved in a scene file.

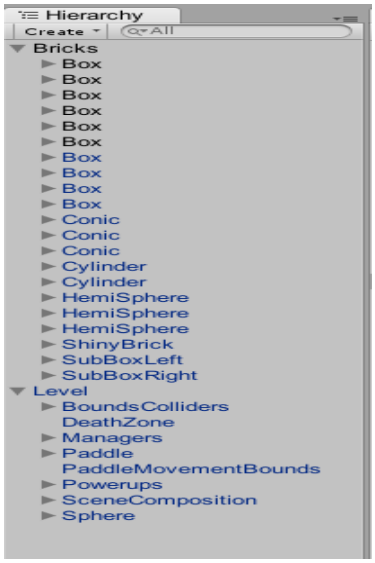

Figure 3: Screenshot of the hierarchy view. Screenshot [3].

Shown in figure 3 is the hierarchy view where are all the objects that make up the first level of the brick breaker game. Any entry which has an arrow next to it can be expanded to show more objects; the arrow indicates a group of objects. This hierarchy view is extremely helpful when there are many objects in a scene and just one of them needs to be found in the scene. The object can be double clicked in this view and it will be selected and zoomed in on the scene view.

2.4 The Project View

The project view is where all the scripts and scenes are accessible from. This view is exactly like the file explorer on Windows or Mac and allows creating files and folders to help organize the projects assets.

Figure 4: Screenshot of the project view with a list of all the assets for brick breaker. Screenshot [3].

Figure 4 shows the project view with a list of all the assets in the project for one level of brick breaker. Most levels will share the same set of assets, scripts and scenes. The scenes will be different depending on the level of the game. For this project it was decided to create a folder for scenes, scripts, sounds, assets and textures. This is to help keep assets separate from each other and make it easier to find assets for one particular game if needed.

2.5 The Inspector View

The inspector view is where all the physics and properties of the objects are stored and accessed from. Every game object has a transform; this is what holds properties of the object such as rotation, position and scale. Other properties are the physics affecting the object, textures to load on the object and sound.

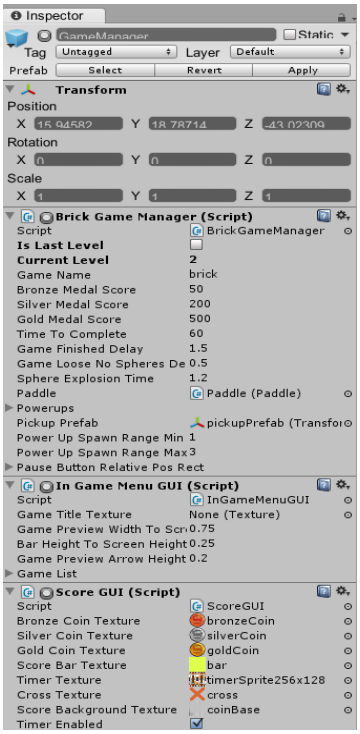

Figure 5: Screenshot of the inspector view showing properties of the game manager object. Screenshot [3].

The properties of the game manager for brick breaker are shown in figure 5. On the game manager there are three scripts (there may be more, further down in the view not shown) and the game managers position, rotation and scale. Loaded on each script is any game object, script or image it will need to load or act upon during the game.

2.6 The Physic Engine

Unity contains the powerful NVIDIA® PhysX® Physics Engine. With this physic engine that is built into Unity many powerful things can be added to a game. Below is a list of several of the features usable in Unity. [3]

- Interactive cloth
- Skinned cloth
- Dedicated wheel collider
- Ragdoll wizard
- Hinges
- Springs
- Character limbs
- Fully customizable configurable joints.
- Soft bodies (like a deflated beach ball)
- Ridged body (which receives forces and torque to make the objects move in a realistic way with no scripting required.)

The physics engine is a great asset with all these features already built in. Without this a game programmer would need to create their own physics engine. Making a physics engine is an entire project in itself. Unity is truly an all-in-one game engine designed to create the next great game.

Every game made during the thesis project uses some elements of this physic engine. For collision detection at least one of the two objects colliding needs a rigid body added to it. The blocks and ball in brick breaker use the ridged body so that they can use to force of gravity to fall or stay on the floor of the level. Physics is a very important part in a game that is being designed today.

3 Common Game Mechanics

Every game that is played today is composed of some very common game mechanics: path finding, collision detection and input. These game mechanics have been around now for decades and have been improved on throughout the years. Here a few very common game mechanics are explained in more detail and how they pertain to the thesis project.

3.1 Collision Detection

The simplest definition of collision detection in relation to games is to determine if two rectangles in the same 2D or 3D space are overlapping. For simplicity 2D space will be discussed in this report. In a 2D world there is an x axis and a y axis. (0, 0) is a point at the top left of the screen. The screen is divided up according to the Cartesian coordinates, so (1, 0) is right of (0, 0) in the x direction and (0, 1) is down from (0, 0) in the y direction. The easiest method of collision detection uses a contradiction method to determine if the rectangles are colliding since it is simpler to calculate if the rectangles are not intersecting. [4]

In bound box testing the rectangles are bound by their left, right, bottom and top edges. To find out if any two rectangles are colliding one simple needs to test for any of the following conditions:

- Rectangle 1′s bottom edge is higher than Rectangle 2′s top edge.
- Rectangle 1′s top edge is lower than Rectangle 2′s bottom edge.
- Rectangle 1′s left edge is to the right of Rectangle 2′s right edge.
- Rectangle 1′s right edge is to the left of Rectangle 2′s left edge.

If any of these conditions are met then the two rectangles are colliding and the game can handle the collision the way it was designed to. There are other methods to determine collisions of objects; circle bound (the same idea as above but using circles, it is less precise) and line to line intersection (used to see if two lines in space intersect). The determining factors for what method of collision detection to use depending on the game design and precision of the collision data needed.

In Unity there is a method already created to help any game enthusiast create a game involving collision detection. The method is called *OnCollisionEnter (Collision)*. In order for this method to work both objects in Unity need to have either a collider attached or a rigid body on the object. The method reports information such as points of contact, velocities, impact speeds and many other collision statistics. Every game designed during

the thesis work project uses collision detection in some form. For instance brick breaker's ball collides with walls and bounces back.

3.2 Finite State Machine

A finite state machine at the simplest form is a model of how a system or a game will behave. Depending on the input from the player the state of the game can change. Each of the games described in the thesis project use a finite state machine to some extent.

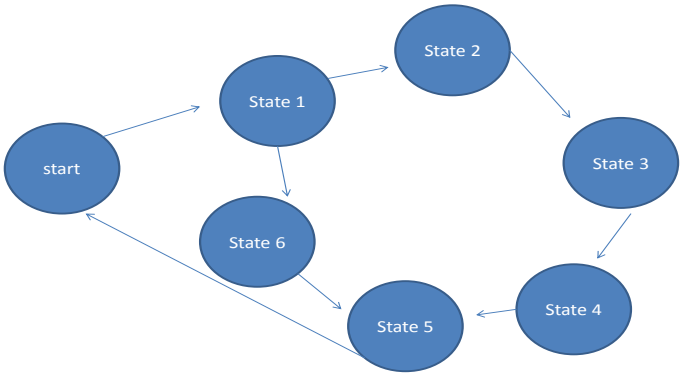

Figure 6: Screenshot from Power Point of a simple state machine with six states.

Figure 6 gives an example of a small finite state machine. From start a change will move to state 1, in state 1 there are two options and depending on which is chosen the state can change to state 6 or state 2. It can continue through its states until it eventually gets back to the start, where once again it can be restarted depending on the input given to the machine. The transition from one state to another is governed by rules. Until the rule is satisfied the state will not change and could even stay for an infinite amount of time. Most games run in a loop waiting for action from the user. This loop that it waits in is more or less infinite and part of a finite state machine and will not change until an action is received.

For the thesis project a small finite state machine was designed within the global game manager. The finite state machine uses the enumeration method in C# to handle the different states and was set up like this: *public enum GameState {Pregame, Running, Paused, Over}.* There are four states the game could be in. "Pregame" is a state where all the objects in the level are loaded into the scene and everything is initialized, it is

also the state the game is in when it is in the main menu. “Running” is the main game loop, as long as a player is playing on a level the game is in the “Running” state. The “Paused” state is for when a player presses the pause button and the pause menu comes up. “Over” is the final state a game could have. The “Over” state is achieved when all the objectives on the level have either been met or the player loses.

3.3 Timers

There are many uses for timers in a game. It would be very surprising to find a game that does not use a timer or time in some fashion. There are countdown timers, count up timers, cool down timers, duration timers and timer based score. [5] In several of the games designed during this project timers were used. In the “Flips” game (from the thesis work project) there is a timer at the beginning to give a preview of the cards. One is also used for flipping the cards to give them a more natural flipping motion. In the “Flight” game (from the thesis work project) one is used for a cool down on the super weapon. The “Drops” game (from the thesis work project) is based entirely on a timer. Once the time in the hour glass is gone the level ends.

In Unity a timer is constructed by using a local or global variable set to the desired time in seconds. Then just subtract *Time.deltaTime* from the variable which will decrease it by 1 second. In early games time was based on frames. Because computers and video cards were slow in the days of DOS (the first Windows operating system) games they could only reach a maximum of sixty frames per second. So the game would just check how many frames had gone by and deduct a second every sixty frames. If this same game was played on a current desktop or laptop computer it would not behave the same way it did so many years ago. Everything in the game would go extremely fast. This is due to the fact that today’s desktop computer can achieve frame rates of over one thousand frames per second. So a new method of time measurement in games had to be developed. Delta time is that measurement and is what was used in any timer made for this project.

3.4 Path Finding

The general definition of path finding is plotting a path from a start point to an end point, done by a computer program or algorithm which is applied to a graph. In many cases the shortest path is the subject of interest to find. In the case of video games it is the same except it is done for a character or group of troops and it plots a path around obstacles on a map.

Real time strategy games are where this is most commonly found. The player builds an army and advances them towards the enemy base somewhere across the map. This map is filled with open space, forests, mountains and a myriad of other obstacles the path finding needs to work around. First person shooters also use path finding with the non player characters and their artificial intelligence would include a path finding algorithm which would be base off a more closed map with defined points.

Here in section 3.4 a few of the most popular, most used algorithms will be discussed and how they are used in both games and other applications.

3.4.1 Dijkstra's Algorithm

Dijkstra's algorithm is one of the most popular path finding algorithms out in the world today. What makes it so popular is that it is very efficient at not only finding the path from one point to another; it also finds the shortest path as it finds its way to the end point. Dijkstra's algorithm is a graph based path finding algorithm. It works by starting with a start node and a queue of adjacent nodes. The adjacent nodes are added to the queue with every iteration of Dijkstra's algorithm and each node in the queue is examined. If it has the lowest distance to the next node it is marked, then all its adjacent nodes are added to the queue. This process is repeated until the destination node is reached, at the same time marking the shortest path on its way. [6]

Dijkstra's algorithm is most used in networking; routers use it to find the shortest path from a computer to a web address the web browser is searching for. It builds a list of hops it needs to take to get to the final address of the web address. This is done because each hop to the next router is given a weight, and so it finds the route with the lowest weight cost and uses that for the route. This algorithm has also been used in games many years ago; it has been replaced by a more efficient version of this algorithm, the A* (pronounces A star). [7]

3.4.2 A* Search Algorithm

The A* algorithm is a spitting image of the Dijkstra's algorithm and works exactly the same except it adds an approximate distance to the end node as part of the weight system. This approximation is done by *heuristic* (a method where trading optimality, accuracy or completeness for speed when the tradition problem solving methods do not work). Using this method allows the algorithm to eliminate longer paths based off this approximation, in turn speeding up the resolution of the shortest path. Using this heuristic approach makes this algorithm faster than the Dijkstra's algorithm. The side

effects of this approach are as the value of the heuristic increases A* examines fewer paths but does not guaranty an optimal path. For video games this compromise is not a problem and is why this is used over just the Dijkstra's algorithm. Whereas in routing protocols a guaranteed path is needed and the A* algorithm is not used. [7]

3.4.3 D* Search Algorithm

The D* (pronoun D star) is an incremental search based on one of these three algorithms, the original D*, focused D* or D* Lite. The original D* algorithm was written by Anthony Stentz in 1994, it was based off of the A* algorithm except that the segment cost can change as the algorithm runs. [8]

The D* algorithm makes assumptions about the unknown terrain, for instance it assumes there are no obstacles within the path from the start to end. It then sets out on the path if it encounters an obstacle, the information in the queue changes and the algorithm is rerun. Similar assumptions about the unknown terrain between its current position and the end destination are made. This process is repeated until the end is reached and a path is made. Working in this way the algorithm runs faster than the A* due to the assumptions made off the heuristic data. Recalling the algorithm is still faster than the A* running once to completion. [9]

D* algorithm is used mostly in robotic application where a robot is to traverse unknown terrain to a destination. It is also widely used in autonomous vehicle navigation systems. These systems are based off the D* Lite algorithm, one such system is a prototype system tested on the Mars rovers *Opportunity* and *Spirit*. [9]

4 Games Developed in Unity

4.1 Brick Breaker

Brick breaker was the first game started for this project. After several long discussions about how the game should work, brick breaker was decided that the game would be a blend of two games; the traditional brick breaker game and a twist on Tetris. For the twist from Tetris it was decided that blocks were built into shapes such as a building. When the lower block is destroyed then the higher block would drop down. To pass the level the player would need to destroy all the blocks which in turn would destroy the building in the level.

4.1.1 Brick Breaker Game Mechanics

There are many game mechanics used in the basic brick breaker game: there is collision detection, scoring, object destruction, power ups, a death zone, touch input and many others. The main idea of the game is that a ball bounces of a pad at the bottom of the screen and then hits bricks which then disappear. Once all the bricks are destroyed the level ends and the player moves on to the next level which would be more difficult than the previous one. In brick breaker the pad is the object on which the sphere is going to bounce and the only object the player can actually control.

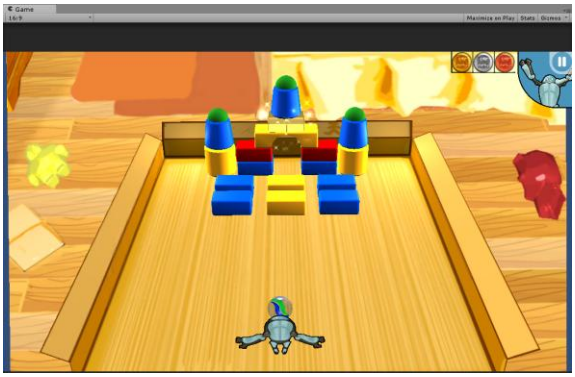

Figure 7: Screenshot of a level from the brick breaker game. Screenshot [3].

Figure 7 shows a level from the brick breaker game, where the robot is used to bounce the ball back and forth to destroy the bricks.

The control settings were defined for the mouse in the beginning, since the games in this project were to be deployed to mobile devices. It also needed to have the input created for touch devices.

4.1.2 Touch Input

To make the game work on all mobile devices a standard input class needed to be created for all games to use. In the class it needed to handle several different cases, one for the mouse, one for Android, one for iOS and one for Windows 8. Luckily Unity has created some pre-processor directives (this allows the script to be partition to compile and execute particular code for specific supported platforms), the ones used in this project are UNITY_IPHONE and UNITY_ANDROID. Due to a lack of testing devices Windows 8 pre-processor directives were not included.

```
#if UNITY_ANDROID || UNITY_IOS
acceleration = Input.acceleration;
//position
if (Input.touchCount==1) {
cursorPosition = Input.GetTouch(0).position;
}
//ignore clicks on marked rect (for GUI)
if(!ignoreButtonsRect.Contains(cursorPosition)) {
//button 1
if(Input.touchCount == 0) {
isButtonUp = true;
isButtonDown =        false;
}
if(Input.touchCount == 1) {
if(isButtonUp) isButtonDown = true;
isButtonUp = false;
}
//button 2
if(Input.touchCount == 2) {
if(isSecondButtonUp) isSecondButtonDown = true;
isSecondButtonUp = false;
#endif
```

Listing 1 pre-processor directives for Unity show in Mono Develop. Listing [10].

Here listing 1 demonstrates an example of the pre-processor directives used to get the touch input working in this project. The first line is the actual declaration of the pre-processor directives that were used. The double bar "||" in programming language is an "and" so in one line it is declaring to use both Android and iOS pre-processor directives. Unity has an Input class which is called with Input.method where method is any

member method in the input class. The script in listing 1 is demonstrating how to check what input method is being used and then handling that specific input. During the development phase most testing was done with the Unity editor.

To make certain that the touch input was working correctly, it was deployed to several Android devices and two iPads. Also in listing 1 is an example of a different way to check for the different ways touch is handled on the different devices. The first example is Input.touchCount; this method is used to check how many fingers are touching the screen. The Input.getTouch(0) is the method for getting information about the first touch to the screen. As the index which is 0 is increased it would give information about the second touching finger and so on.

As listing 1 also shows the input is used for both button operations from the mouse. The script is checking to see if the right or left mouse button is being pressed, in the script the left mouse button is “button” and the right mouse button is “secondbutton”. The final thing to remember with the pre-processor directives is to end them with the “#endif” statement. If this is missing the script will not compile. Any script within the “#if” and “#endif” statements will only be compiled when the Android or iOS platform is selected in the build window in Unity. Otherwise this code does nothing when it is compiled for the computers.

4.2 Flips

The Flips game is a basic card flipping memory game. On the cards are the images of all the game characters used throughout the game. In the “Flips” game no special features of the mobile devices are used except for the standard touch input that all the games use. Development of a child’s memory is very important, children being able to recognize and match similar images is a great game and an addition to the lineup of smaller games included in this project.

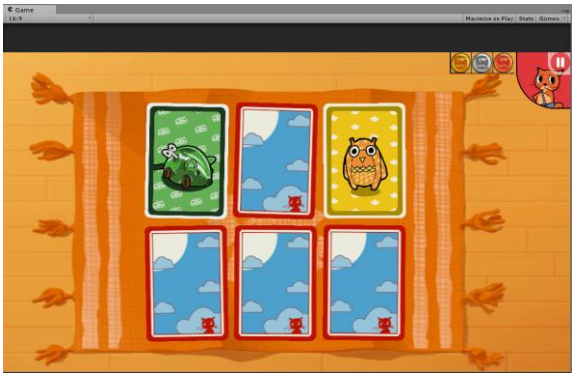

Figure 8: Screenshot of the flips game showing two non matching cards flipped over. Screenshot [3].

Flips Game Mechanics

As stated in the previous section there is nothing new in this game as far as features that does not mean there are no notable game mechanics to discuss. The whole idea of the game is to touch a card, it flips over and touch another card as can be seen in figure 8. If they both match they disappear and when the board is cleared the level ends. Flipping the card over in a timely matter is the real trick in this game. The cards are made up of two one sided planes.

4.3 Mouse in a Maze

The mouse in a maze is a classic game of a little mouse running around in a maze. The game was designed to be used with the mouse when on a computer and the accelerometer when used on a mobile device. This game uses many of the game mechanics that were previously used in the Brick breaker game such as collision detection, power ups and a death zone.

Figure 9: Screenshot of level one of the maze game. Screenshot [3].

Figure 9 shows the first level of the maze game. The player needs to direct the mouse through the maze and collect the cookies as he/she moves about the maze. The main

goal of creating this game was to learn how to use the accelerometer on the mobile devices and then add this into the tutorial that was also a part of this project.

Mouse Maze Game Mechanics

For the mouse in a maze game, the main game mechanics that was used was to teach how to use the accelerometer on the mobile devices. As in the brick breaker game this was done using pre-processor directives so that it was only checked if the code was compiled for a mobile device.

```
public class InputManager : MonoBehaviour {

Vector3 acceleration;
Vector2 cursorPosition;
Vector3 accel = new Vector3(0,0,0);

#if UNITY_ANDROID || UNITY_IOS
acceleration = Input.acceleration;

#else
accel = new Vector3(0,0,0);
if (Input.GetKey(KeyCode.UpArrow)) {
accel += new Vector3(0,0.5f,0);
} else if (Input.GetKey(KeyCode.DownArrow)) {
accel -= new Vector3(0,0.5f,0);
}
if (Input.GetKey(KeyCode.LeftArrow)) {
accel -= new Vector3(0.5f,0,0);
} else if (Input.GetKey(KeyCode.RightArrow)) {
accel += new Vector3(0.5f,0,0);
}

acceleration = accel;
cursorPosition = Input.mousePosition;
if(!ignoreButtonsRect.Contains(cursorPosition)) {
isButtonDown = Input.GetMouseButtonDown(0);
isSecondButtonDown = isButtonDown;
}
```

```
else print ("in rect");
#endif

acceleration.x = Mathf.Clamp(acceleration.x, -0.5f,
0.5f) * sensitivity;
acceleration.y = Mathf.Clamp(acceleration.y, -0.5f,
0.5f) * sensitivity;
}
```

Listing 2: Code snippet for detecting accelerometer movement in mobile devices. Listing [10]

Listing 2 examines a code snippet where the pre-processor directives are used again. This time it is for retrieving information from the accelerometer and using it to direct a mouse that is running around in a maze looking for cheese and cookies while trying not to run into any traps.

To get the information from the device the method Input.acceleration is used from Unity. The output of this method is set to a local variable. The local variable is used to change the mouse' position in the direction taken from the accelerometer after the "#endif" statement. Inside the "#if" statement if the code is compiled for a device, then the accelerometer is used; otherwise, the arrow keys on the keyboard are used.

To get which key is pressed the Input.GetKey(KeyCode.keypressed) method is used. As shown in listing 2, all the arrow keys are checked and if one is pressed then another local variable is changed and then used to set the direction of the mouse on the screen. In this game a few other game mechanics were also used. For instance teleporting, slowing effects and spinning when different terrain was encountered.

4.4 Flight Game

Flight game is a classic side scrolling shooter game. Again many of the same game mechanics are used here. The special aspect of this game that was implemented was a newer touch input method seen in many first person shooters and flying games on mobile devices. This is where in the lower left corner of the screen is a small solid circle. This small circle is encased in a larger circle. The smaller circle is dragged from the center in the direction the player wants to go. In some games the further from the cen-

ter the circle is the faster the player's character will go. Basically it is a digital joystick on the screen.

Figure 10: Screenshot of the digital joystick used for controlling the dragon. Screenshot [3].

Here in figure 10 at the bottom left is the digital joystick created for this game and used to control where the dragon flies. Also on the right hand side, are two buttons for the different attacks the dragon has for defeating the enemies flying at him. This is a newer approach to input from a player, for these types of games. This however does limit the play area of the game a bit and needs to be considered when using it in games. Many players do not like this type of input as it obstructs the view of the game.

4.5 Candy Monster

Candy monster is a newer game type that can be found in different variation in any of the app stores for the devices. It is a game that gets the player to use logic, deductive reasoning and physics to help get the candy to the little wizard. In the first few levels the object is to use what is on the level already, as well as draw a line for the candy to fall on and direct it to the wizard. There are stars in each level that the player can also try to collect by having the candy roll over it.

Figure 11: Screenshot of the first level in Candy wizard game in Unity. Screenshot [3].

Figure 11 illustrates what the player is presented with when they select the candy wizard game. The object is to get the candy to the wizard by drawing a surface it can roll on.

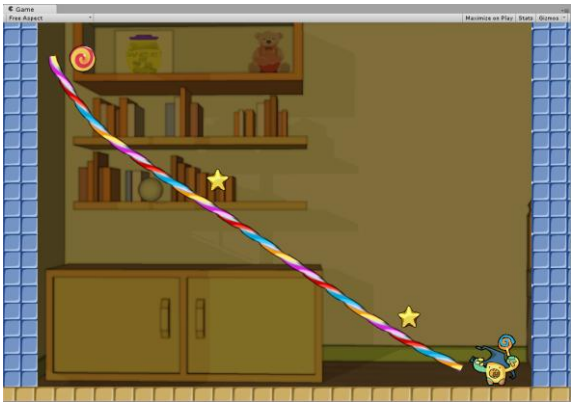

Figure 12: Screenshot of the solution to the first level in Candy wizard. Screenshot [3].

Here in figure 12 the solution is shown to the first level of the Candy wizard. The player just needs to draw a line from the candy to the wizard that will also allow for the candy to roll over the stars to collect them. In later levels it becomes more complicated as there is also a speed potion to be used. The player can put the speed potion onto specific areas of the line he/she has drawn in order to help the candy get up and over to the wizard.

Figure 13: Screenshot of a level using the speed potion. Screenshot [3].

Shown in figure 13 the blue glow around the lines illustrates where the speed potion is used to help accelerate the candy so that it can get to the wizard while also collecting the stars of that level. Also shown is the eraser icon that can be used to erase any of the lines drawn that were wrongly placed on the screen.

4.6 Grabs

The idea of the Grabs is that the robot character is helping to clean up the room and throwing all the toys that are on the ground up in the air and the doll is running back and forth across the screen to catch them and put them away.

Figure 14: Screenshot of the doll trying to catch falling toys. Screenshot [3].

Shown in figure 14, the doll is trying to catch a falling teddy bear. The iron falling is not something the doll should catch as its heavy and could hurt her. There is really just collision detection and touch input used by this game. In the top right corner is the hour glass mentioned earlier in the timer section. This is the only game where the timer is actually visible to the player.

5 Deploying Games to Mobile Devices

Since this project is developing these games for mobile devices there are downloadable files required to deploy the games. The SDK's for Android needs to be downloaded, Xcode for Apple devices, and the Windows SDK for Windows 8 devices. As for Windows, Linux and Mac there is nothing more needed to add, Unity does all the required compiling for these platforms.

5.1 Deployment to Android Devices

For Android the Android SDK is needed. The most current one can be found by doing a Google search. Once the SDK download is complete the Update Manager should be opened and any packages that are out of date should be updated. Once all updates are finished the path to the SDK should be added to Unity's build properties. Select Edit > Preferences.

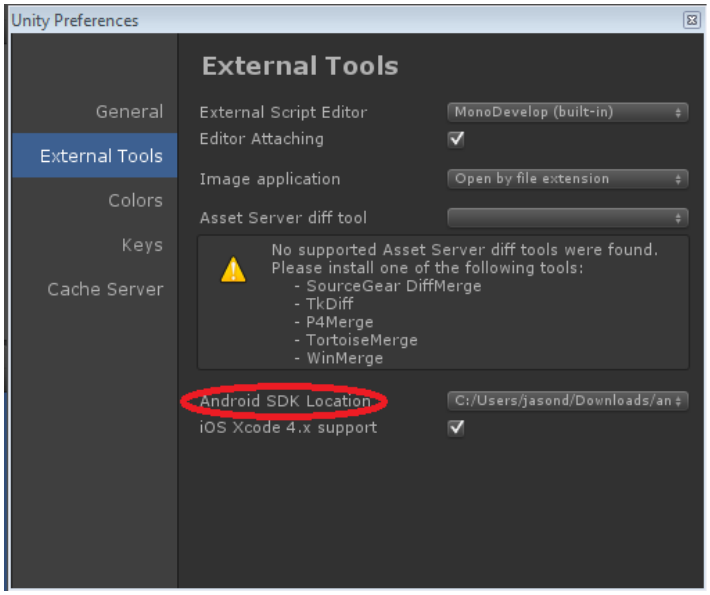

Figure 15: Screenshot of the Unity preference window. Screenshot [3].

Here in figure 15 the Android SDK location is highlighted red. Set the location by clicking the path and selecting where the SDK is saved on the computer. Attach the Android device in with the USB cable, allow the device to run in debug mode and it should be ready to run. If nothing comes up to allow to set it up in debug mode then on the device navigate to Settings > Applications >Development, within this screen tick the box next to USB Debugging as well as Allow mock locations.

One other item to address is the drivers for the device. If they are not installed Google to search for them. As part of the thesis project it was a goal to have this game work on

multiple mobile platforms. To this end the game was built many times and tested on several different Android devices. Both phones and tablets were used in the testing process.

5.2 Deployment to Apple Devices

For deployment on Mac and Apple devices the process is much more detailed. The first thing that is needed is a Mac with Xcode running on it. With this done the game can be ran on the Mac and in the device emulator for all of the Apple mobile devices. To be able to deploy the game to a device an Apple developer license must be purchased from Apples' web site.

Once the license is obtained the device where the game is to be deployed needs to be provisioned. On the Mac open Xcode and provision the device to make it build. In Xcode navigate to Window > Organizer. In the Organizer select Devices from the several icons across the top. At this point attach the device to develop on. When it's plugged in the name of the device should show up on the left hand side of the screen. Select the device name now and it should display some info.

On the bottom right of the screen there should be a button that says Add to Portal. Click this button and sign in using an Apple ID and password. This will take and add the device to the portal; this downloads all the required certificates and profiles on to the Mac as well as installs them to the device. Once this process completes you go to Unity select File > Build Setting, select iOS, click the switch platforms button at the bottom left of the window and finally build and run. It will build in Unity, open and run in Xcode and then deploy to the device connected to the Mac.

5.3 Deployment as a Cross-platform Game

The great benefit of using Unity to create games is that it has been built to deploy games to many different platforms. It itself is cross-platform, which means unity will run under Mac, Windows and Linux. Since Unity runs on all these platforms, it too can build the game for each of these platforms. Also in the list of platforms it can build for are: PlayStation 3, Xbox 360 and Nintendo Wii, each needing a separate purchasable license and possible developer's license from the respected companies. For this project the game was built and tested in both Windows and Mac. Unity will not build these two in the opposing operating systems. For example Unity will not build the Mac version of the game while running in Windows. To build the game for a specific operating system or mobile device in Unity navigate to File > Build Settings.

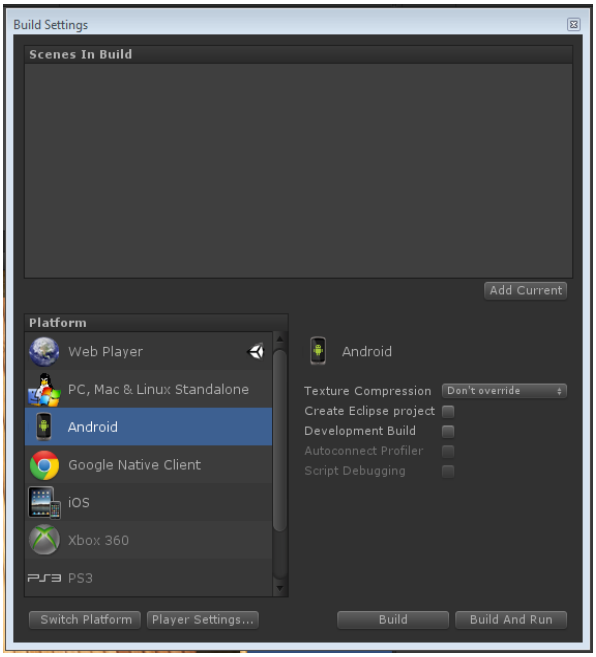

Figure 16: Screenshot of the Build settings window in Unity. Screenshot [3].

Here in figure 16 the build window is shown with Android currently selected. To select a different platform, highlight the preferred platform and the click the Switch Platform button. This process can take some time depending on how big the game is and how fast the system is that the game is running on. It needs to import all of the assets from the game to the chosen platform before it can create a build for that platform.

6 Writing Technical Training Material

Another aspect of the thesis project was to write a tutorial that would walk a student through the process of creating this game. One problem that arose was how the tutorial should be written. Whether it is done like most tutorials where it starts out at zero and explains from start to finish how to reproduce the game. Or should it emphasize the important topics that should be learned throughout the tutorial, but not be a complete walk through which ends with a complete game? After several meetings and smaller tutorials had been, written the decision was made to have it be more of a hybrid tutorial. It would start from scratch; for example it would show the user how each component of Unity worked, how a state machine works, what the game manager does and how to use collision detection.

As the tutorial progressed it would then highlight the important game mechanics that was built into each of the smaller separate games. For instance: using the accelerometer in mobile devices, how to use touch input instead of a mouse and using more advanced physics in the games.

6.1 Graphical User Interface (GUI)

One very important topic it would also cover is the graphical user interface. This is a major part of the game. If its design is poor it could take a great game and make it average. The team took several hours to do research on what type of menu the game should have. Once each member had an idea of the type of menu they thought the game should have a long debate took place to decide which one would be the best. As this was very important to the success of the project a great deal of time was dedicated to this task.

Figure 17: Screenshot of the Toy Party main screen. Screenshot [3].

In the end it was decided that the menu would start off with six buttons at the top, the main game image, an exit button on the left and a credit button on the right as shown in figure 17 above.

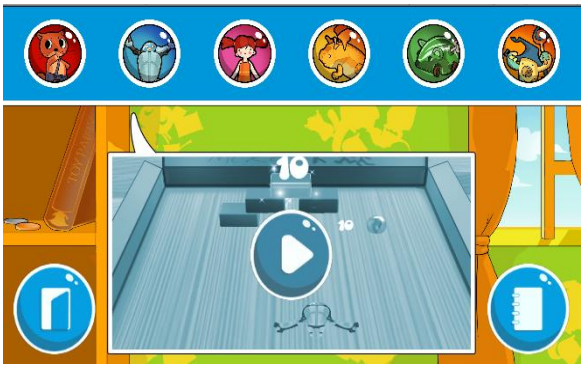

Figure 18: Screenshot of the Toy Party menu with brick breaker selected. Screenshot [3].

As seen in figure 18 when a game button is pushed a box will appear in the middle of the screen with a picture of the game selected and have a play button on it. If the play button is pushed, it would continue to the tutorial screen.

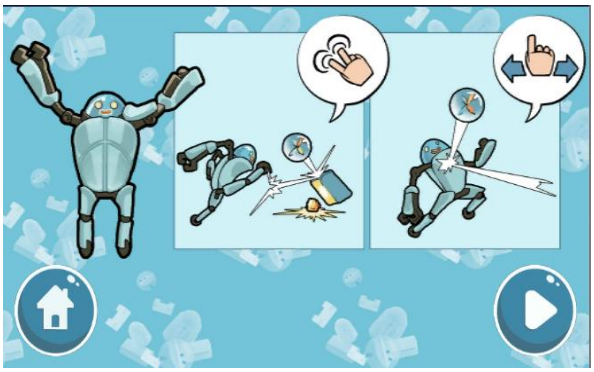

Figure 19: Screenshot of the Toy Party tutorial screen for brick breaker. Screenshot [3].

Figure 19 is a depiction of the tutorial for the brick breaker game. The two boxes have images of how the game should be played. Once the player understands what to do, they can press the play button in the right corner. If they decide not to play the game they can get back to the main menu buy pressing the home button in the left corner.

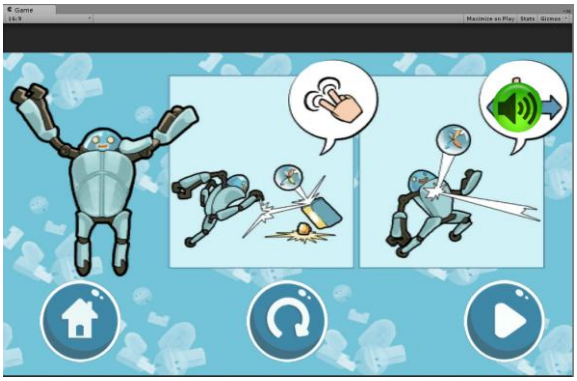

Figure 20: Screenshot of the pause screen for the brick breaker game. Screenshot [3].

Once in the game, there is a button at the top right corner to pause the game. If this button is pressed then the pause menu would open. As shown in figure 20 the background of the pause menu is the tutorial screen. There are four buttons on the screen: in the bottom left corner is the home button, a restart button in the middle, the resume/play button in the bottom right corner and finally a sound button in the top right corner so the sound can be turned on or off without going to the main menu and accessing the options menu.

Figure 21: Screenshot of the win/lose screen for brick breaker. Screenshot [3].

The final menu screen is the win/lose screen. In figure 21 at the end of the game the player is presented with a screen to inform them if they have won the game and if so what medal they have earned. Along the bottom of the screen again there are three buttons, the home button in the left corner, the restart button in the middle and the next level button in the bottom right corner. If the player failed to meet the objectives of the game the screen would show a frowning player with a “try again” caption. When on this screen, the next level button is grayed out, not allowing progression to the next level until the player receives a medal from the game.

6.2 The Scoring System

Another area of heated dispute was how the scoring in the games would work. At first each game had its own scoring method, yet at the end of each game the player would gain a medal either off the points tallied, lives left and items picked up and so on. This brought up inconsistencies in the overall game and at the end of each game.

It was finally agreed that all the games would build a score from points earned in the game and that it would take a certain amount of points to gain a medal. If not enough points were earned in the level to earn the lowest medal, then the player would have to repeat the level over until they could attain the bronze medal. After achieving this goal, they could them move on to the next level of that game.

If the player loses a life during the level, then their chances to gain the highest medal is lost. If they lose another life, then they could no longer gain the silver medal. If all three lives were lost, they would need to start the level all over again because all chances to gain a medal where lost. With every game scoring this way, the player could change from one game to another and not worry about how to win at each game. Instead they could just learn to play the game well according to the one scoring method.

6.3 Main Game Character

In many games that are played these days there is a character who a player would play and help through the game or a character that would show up and encourage and help the player through the game. When three different games were designed and working in the thesis project the question of should there be a main character in this game came up.

The group members deliberated for several hours and came to a decision that each game would have its own character that the player either controlled or would help the character out in the game. So for the brick breaker game a robot was chosen, for the maze game it had to be a mouse, for the flips game a cat, for the flying game a dragon and for the candy wizard a monster.

With the important choice of the characters for the game made, it was decided that the main screen of the game would consist of all the characters on a table in a little kid's room. It was also decided that all the games would happen in the room and would have a theme that deals with different activities that little boys and girls would do in their

rooms. With the characters chosen it also lead to the images that would be used for buttons of each game, the button would show the hero of the game on it.

7 Game Testing Results

The project was tested at several different times during its production. Once the team had a demo version where all six mini games where together in one large game and all the GUI screens were connected correctly, the game was tested by children between the ages of four and twelve. It also was given to adults to get their point of view on the game and how it progressed over time. The results of the testing are described below.

7.1 A Child's Point of View

The game was deployed to two iPads and then tested by about ten different children throughout different development phases. It was not until all six games where combined together into one collective game that any testing was done with children.

Once in the hands of children the team could see what worked and did not work. It was a good test for the GUI. In most cases the children could easily navigate the different menus and get to the games they found most fun. It was observed that children five and under got lost after about the second touch through the GUI. The child would first pick a game from the top by tapping one of the six game icons; this would open a window in the middle of the screen which had a play button. The child then needed to tap the play button which leads to the tutorial page with a final play button they needed to press to get the game started.

It was found that this tutorial page was a bit confusing for most players. They often asked how to start the game once they got to the tutorial page. This issue was brought up to the project manager but it was decided the tutorial page would stay despite its confusion to players.

The first test run proved to be successful as all of the children asked if there were more levels to play in each game. As the games developed, more levels were added and the difficulty increased, the test subjects became more and more involved in the game. The most popular games were the brick breaker and the candy wizard especially with children older than eight. The younger children generally would play the games which had a simpler game play and did not get to difficult with each level the flips game for example was a popular one with ages from four to six.

7.2 An Adults Perspective

The feedback given by the adults that played the games was completely different. The adults were also confused once they got to the tutorial screen for the chosen game.

Other observations by the adults consisted of aesthetics in the menu, colors and sound. In their opinion in the first test build the colors were low in contrast and brightness, and on the iPad it looked very dull if compared to other popular games.

With the feedback the colors where brightened and given a high contrast so that they stood out. Another item in the feedback was that the game lacked levels; this was expected as this was more of a demo with one or two levels per game. After several releases each game had a minimum of five levels where the difficulty was increased with each level.

With the newer releases came more feedback, mostly very positive. In one release the feedback was that the difficulty in several of the games grew to be too hard to fast. Through the entire testing phase this was a common problem. As the programmers developed the new levels they got used to playing the games and ended up making the next levels more difficult for them and not the audience that the games was originally designed for, which was children four to eight years old.

In earlier test builds no sound was a big complaint by the adults that played the game. This problem was also addressed and generally with each new test build one additional game would have sound effects and music. It was surprising to find that no sound was a big issue for nearly all the adults who played the game.

When the game was nearing a stage of completion students from an affiliated college in Korea came to visit the project and see the progress made. These students were very impressed with the near final version of the game. Their feedback to the game was that it could still use more levels, as at this stage each game had five levels. The games which were made first had ten levels. This gave a good indication the games were good and people in general would want to continue playing if there were many levels to complete.

8 Game Addiction

In our modern society computers are everywhere, if it is not a computer it is a laptop, tablet or smart phone. It is reported that ninety percent of homes have a computer and of these homes eighty percent have a high speed internet connection. [11] There is no doubt that children are using computers in many ways for instance: for homework, learning, reading, typing and of course playing video games. One major problem associated with computer usage and games is video game addiction.

Wikipedia defines video game addiction in the following way: it is “an excessive or compulsive use of computer games or video games, which interferes with a person's everyday life. Video game addiction may present as compulsive game-playing; social isolation; mood swings; diminished imagination; and hyper-focus on in-game achievements, to the exclusion of other life events. In May 2013, video game addiction was added to the *Diagnostic and Statistical Manual of Mental Disorders* in the Conditions for Further Study section as "Internet Gaming Disorder”. [12]

It is not uncommon to find children playing their favorite video game for hours each day. Knowing the key signs that would identify a child as a game addict is especially important. For example, CRC health group claim that: “that 10 percent to 15 percent of gamers exhibit signs that meet the World Health Organization’s criteria for addiction. Just like gambling and other compulsive behaviors, teens can become so enthralled in the fantasy world of gaming that they neglect their family, friends, work, and school.” [12] Excessive game playing can lead to behaviors and symptoms which resemble many other physiological disorders and even symptoms of a drug addict. These include:

- Isolation
- Neglect of personal hygiene
- Preoccupation
- Sleep deprivation
- Lack of control
- Skipping meals
- Loss of time
- Lack of development in other areas of life

There are many other symptoms and the impact that game addiction has on people is alarming.

Game addiction is not yet officially recognized as a diagnosable disorder by the American Medical Association. Game addiction is a serious problem facing our teens and pre-teens in this computer age. Here are a few extreme cases where game addiction has lead to loss of life in the real world, often from exhaustion, loss of time or loss of attention to real world situation:

“In 2005, Seungseob Lee (Hangul: 이승섭) visited an Internet cafe in the city of Taegu and played *StarCraft* almost continuously for fifty hours. He went into cardiac arrest, and died at a local hospital.” [12]

“In 2009, Kim a-rang, a 3-month-old Korean child, died from malnutrition after both her parents spent hours each day in an internet cafe raising a virtual child in an online game.” [12]

These are extreme cases and not all gamers are addicted. Many children, teens and adults can play video games a few hours a week and still balance school, work, friends and life in general.

9 Violence in Games

It is an unfortunate problem that in most video games there is violence depicted. On Amazon.com, a popular video game being sold is titled "Call of Duty 4: Modern Warfare. [13] It is a game where the player chooses to play a solder sent into different important battles of wars fought in the past. The player's goal is to defeat the world's most dangerous enemies. This game has life like images and the violence is very graphic.

As violence has increased in games so has the concern of its affect on those who play the violent games. There have been several mall shootings, school shootings and mass shootings by individuals who have during their investigation made suggestions that video games have been the motivation for their killing sprees. One such case is the Columbine High School massacre in the United State on April 20, 1999, where 13 people were shot dead. Both teenage shooters contributed their motive to do the shootings from games such as *Wolfenstein 3D* and *DOOM*. [14] It is unfortunate that several other shooting of this same caliber have been in some way linked to video games to some extent.

Many researchers argue that the violence in video games desensitizes the player which can affect their feelings towards real life violence. "Several games have garnered significant media attention, including 2004's JFK assassination re-enactment *JFK Reloaded,* 2005's Columbine shooting re-enactment *Super Columbine Massacre RPG!,* and 2006's *RapeLay,* a Japanese video game where the player stalks and rapes a mother and her two daughters." [15] Researchers also argue that violence in video games teach children that the use of violence to solve conflicts is acceptable.

10 Positive Effects of Video Games

With all the research of video games being a bad influence the researchers are now also looking at the positive aspect that games create. Games catch the full attention of the player and immerse them into amazing 3D worlds. These worlds are a perfect place for learning too; games have the ability to actively involve students in learning. Virtual environments are a place where a child can make decisions which may have a terrible outcome. They can see how their decision plays out, if needed go back and refine the decision made to achieve the outcome they were trying for. This teaches that information gained from failing allows for a better chance at future success.

Online role playing games have been found to help players build collaborative skills while completing game objectives. Video games can help with computer literacy, develop dexterity and fine motor skills, memory recall, fact finding, increase self esteem, help making friends, develop skills for the classroom and business world, creative thinking and a wide variety of other skills and social benefits. If there is something a person wants to teach they can construct a game which with the principles they want taught, this will help the intended audience learn the topic.

On the subject of violence in video games “there is a study being conducted by Dr.Cheryl Olson and her team at Massachusetts General Hospital’s (MGH) Center for Mental Health and Media and Harvard to prove that violent games help students deal with stress and aggression. She has found that over 49% of boys and 25% of girls use violent games such as Grand Theft Auto IV as an outlet for their anger. Many authors disagree with the notion that suggests that the media can cause violence, they propose media cannot cause violence because humans have the ability to recognize what is wrong, and what is right. They suggest people are not going to mistake fiction for reality.” [16]

Dr.Cheryl Olson’s study shows that violence may indeed have a positive effect rather than the negative effect most research is focused on. Dr.Cheryl Olson stated “If video games do cause youth to be violent, then one would expect juvenile violent crime to increase as more youth play violent video games. Instead, the arrest rate for juvenile violent crimes has fallen 49.3% between 1995 and 2008, while video game sales have quadrupled in the same period.” [16] It can be concluded that children need an outlet for their anger, video games with violence can be used as a healthy outlet when used in moderation and the children, the games and time used is monitored.

In the thesis project violence in the games were kept to a bare minimum. Three out of the six mini games have some type of violence. The project group made a decision that the violence would not be directed at humans. For instance Brick breaker the robot is being violent in destroying the buildings but no one gets hurt. On the flip side the Flips game was made with the idea in mind to help children to learn pattern recognition. This gave the project a positive aspect for children. During the project both the good and bad effects of the games where considered and change if it created a negative effect.

11 Conclusion

Game development in Finland has been becoming a big enterprise lately, with companies like Rovio, Redlinyx, Remedy and Supercell putting out big name titles and becoming very successful. To develop a game that could even live up to the likes of the games from these game companies, one needs to know the tools used to do so. During the project described in the thesis Unity 3D was used to develop the six games. Unity is also the game engine that several of these game companies are using to produce their top hits. Having a firm foundation in Unity will give any game enthusiast the ability to obtain a job with these fine game companies. This report has covered all the basics of this game engine and the programming language used to create the game mechanics used in top games today.

All games share some very common game mechanics. This report has looked at just a few that a programmer will find in every game produced today from collision detection to path finding to timers and finite state machines. All of these game mechanics are very important to any game. The few mechanics listed here have been covered in detail as they pertain to the games made during this project. Every game produced used at the minimum collision detection. Most also used timers and the finite game machine. Path finding was used in the flying game just a little and was covered as it is a major part of many games produced today, especially first person shooters and real time strategy games.

In this project six games where made as a showcase to get more students interested in the game programming degree offered at Metropolia University of Applied Sciences. Many of the games were built to feature a different aspect of the mobile device such as touch input or using the accelerometer. The six games were made into one large game where different heroes are used in the different games to help the player in the game.

These games were also the subject of training material made for students who will be enrolling into the game programming degree. The tutorial was a hybrid of the most common types of tutorials out today, where the student would learn the basic elements from each game but not make the complete game exactly the way it was done in the project. This allows for the student to take what they have learned and branch out making their own games based off of what was learned in the tutorial.

There is no doubt that nearly every home in a developed country has both a computer and some form of internet. This in itself is not the problem but the time used by adults and children on the internet and computers is. There are many great reasons a child uses a computer: homework, reading, learning, typing and of course playing games. When a person uses the computer for gaming for many hours a day every day it can develop into an addiction.

Knowing the common signs that might suggest one is addicted will help in getting that person help so it does not become an uncontrollable problem. Children can especially be prone to addiction as online games allow them to make friends, socialize, complete achievements, spend endless amount of time inside and idle. With games becoming more lifelike and immersive it becomes harder to get children away from the computer and outside getting the exercise they need.

To this end the burden lies solely on the parent to monitor their children's game use, what games they play and what other activities the child needs to engage in outside of games.

Now-a-days violence is everywhere, on the TV, in movies and in video games. The big question that arises is with all this subjection to violence is what kind of effect it has on the children playing games, watching TV and movies? There have been shootings that have at first been attributed to video game, and in the end of the investigation been cleared as the assailant had other mental issues or life events that lead to them committing the crime.

Research has looked at the negative sides of the influence of violence in games and has concluded that it can lead to desensitization of the players feeling to real life vio-

lence. It has also believed that it teaches children that using violence to resolve issues is an acceptable practice.

The negative effects of games have been well researched, so what about the positive effects a game can have on children. Online role playing games have been found to help players build collaborative skills while completing game objectives. Video games can help with computer literacy, develop dexterity and fine motor skills, memory recall, fact finding, increase self esteem, help making friends, develop skills for the classroom and business world, creative thinking and a wide variety of other skills and social benefits.

This is just a few of the positive effects that games can have on children. Research in to the positive effects of games have only began, as research evolves more on the positive effects on games The balance of good and evil done by games will be better understood and will give parents the ability to choose what a child will get out of the games they play.

Developing video games is no easy task. A game developer must weigh out the pros and cons of violence that is added into the game. The developer must also consider if the game will produce any educational value to its players. The research done for this thesis show that violence can be used as a healthy outlet for aggression and anger in teens. While also giving them a safe environment to learn in and experiment with making choices and seeing how those choices unfold without real consequences.

References

1. de Byl, Penny. Holistic game development with Unity. Focal press; 2012.

2. Goldstone, Will. Unity 3.x game development essentials. Game development with C# and JavaScript. Packet Publishing; 2011.

3. Unity 3D game engine [computer program]. Version 4.3, Unity Technologies: October 15, 2013.

4. devmag.org.za. basic-collision-detection-in-2d-part-1/ [online]. Last modified April 13, 2009
   URL: http://devmag.org.za/2009/04/13/basic-collision-detection-in-2d-part-1/
   Accessed October 15, 2013.

5. Scirra Ltd. Timers [online]. Last modified January 25, 2013
   URL: https://www.scirra.com/tutorials/450/timers
   Accessed October 15, 2013.

6. E. W. Dijkstra. Dijkstra's original paper. A Note on Two Problems in Connection with Graphs. Numerische Mathematik 1, 269 – 271 (1959).

7. Pearl, Judea. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley. October 15, 2013.

8. Stentz, Anthony. The Focussed D* Algorithm for Real-Time Replanning. Proceedings of the International Joint Conference on Artificial Intelligence. October 15, 2013.

9. Wikipedia. D* [online]. Last modified on August 26, 2013.
   URL: http://en.wikipedia.org/wiki/D*.
   Accessed October 15, 2013.

10. MonoDevelop [Computer program]. Version 4.0.1, Open source software: October 15, 2013.

11. Wikipedia. Video game addiction [online]. Last modified on October 13, 2013.
URL: http://en.wikipedia.org/wiki/Video_game_addiction.
Accessed October 15, 2013.

12. CRC Health Group. Video Game Addiction [online].
URL: http://www.video-game-addiction.org.
Accessed on October 15, 2013.

13. CRC Health Group. Violence and Video Games [online].
URL: http://www.video-game-addiction.org/violence.html.
Accessed on October 28, 2013.

14. New York Times. columbine_high_school [online]. April 17, 2008.
URL:http://topics.nytimes.com/top/reference/timestopics/organizations/c/columbine_high_school/index.html.
Accessed on October 28, 2013.

15. ProCon.org. Do violent video games contribute to youth violence [online]. Last updated on October 15, 2013.
URL: http://videogames.procon.org/#Background.
Accessed on October 28, 2013.

16. Wikipedia. Video game controversies [online]. Last modified on 28 October 2013.
URL: http://en.wikipedia.org/wiki/Video_game_controversies.
Accessed on October 28, 2013.