Kinect Game for Children with Intellectual Disability

GRADUATE PROJECT

Submitted to the Faculty of
the Department of Computing Sciences
Texas A&M University-Corpus Christi
Corpus Christi, Texas

In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

By

Rahul Bethi
Summer 2017

Committee Members

Dr. Scott A King
Committee Chair

Dr. Toyin Ajisafe
Committee Co-chair

Dr. Ajay K Katangur
Committee Member

ABSTRACT

Children who suffer from ID (ID) are often significantly less physically active than those who do not. To overcome these disabilities to some extent, children should undergo instruction, structured training, context-specific exposure to skills and practice to acquire loco-motor skills such as jumping, ducking and hopping. It is challenging to encourage children with ID to perform these activities. To overcome this problem, a 2.5-dimensional scroller game using a Microsoft Kinect device is developed. This game exploits the motivating nature of virtual reality to improve children's loco-motor skills while they are at home. A computer game is developed using the Unity game engine and a Microsoft Kinect device. The game is played with basic loco-motor movements performed in front of the Kinect, which captures the players movements by collecting the RGB and depth image data and sending them to the computer. The game also obviates some of the practical difficulties of taking the children to the activity center due to limiting factors such as adverse weather, poor transportation or busyness of parents. This project is intended to help the Department of Health & Kinesiology at Texas A&M University Corpus Christi.

Keywords: Microsoft Kinect, 2.5D scroller game, loco-motor skills, children, intellectual disability, Unity game engine.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

CHAPTER 1

BACKGROUND AND RATIONALE

Children with ID face many difficulties participating in physical activities. Only 39% of children with ID participate in organized sports at least once a week compared with 84% of children without ID [11, 2]. Children with ID face challenges to participate in physical activities and later carry these problems into adulthood [9, 7]. Adequately developed locomotor skills such as jumping, hopping, ducking and running form the foundation for acquiring sport-specific skills and skills to engage in physical activities throughout a persons life. Jumping is integral to perform complex activities in various sports, including volleyball, basketball and soccer; however, people with ID have shown inferior vertical jumping performance compared with peers without ID [5, 6, 1, 3]. Unlike walking, which develops automatically, locomotor skills such as jumping are more often acquired through instruction, structured training, context-specific exposure and practice.

It is relatively easy to convince adults to undergo rigorous training or any specific structured physical activity, but it is more challenging to convince children to follow such instructions, this is the major motivating factor for this project. The project aims to provide a solution in the form of a game to make such activities fun or more appealing for children [8]. These physical activities can be achieved through a game, which can engage the children in performing the desired degree of exercise required. Additionally, taking the children to the activity center can also be difficult for parents for several reasons, such as adverse weather, poor transportation or any other inconvenience, which further highlights the need for a play-centered game that aims to improve locomotor skills.

1.1 Movement of the player

The player (child) who plays the game needs to perform jumping, hopping and ducking in various sequences in order to win. To capture these movements, we must use an image capturing device. Various controllers are used to receive different types of player input. A typical approach is based on motion capture using sensors in which motion is detected by sensors attached at every major joint on the participant's body. A variation of this method is to use small light bulbs in place of sensors and capture those light bulbs using a camera. The captured video is processed to determine where each light bulb is in each frame and the movement of the joints is calculated. However, this is an unfriendly process, especially for children.

Another approach that can be used is based on Virtual Reality (VR). In this method, a device is placed in front of the eyes, attached to the head of the player. It has different sensors to capture the movement orientation and speed of the player's head. Additionally, player can also hold two supplementary controllers in his hands, which can be processed to find out the movements of the player. A smart phone can also be used as a virtual device using special VR glasses, in place of the screen which the VR device normally has. But a smart phone has limited battery and processing power, compared to the capacity of the actual VR device. Oculus Rift is the best example of a VR device which has a screen and all the sensors. However, it is not very comfortable for the children to hold the device or to wear it.

To overcome these limitations the most common approach to capture motion is to use a depth camera. The most commonly used devices use a depth capturing camera alongside game consoles such as the Microsoft Xbox One, Sony PlayStation 4 and Nintendo Wii U. The player stands in front of this depth camera so that the camera can capture and process the player's movement. This project uses Microsoft Kinect

v2[10], which is one of the most efficient depth cameras (Figure 1).

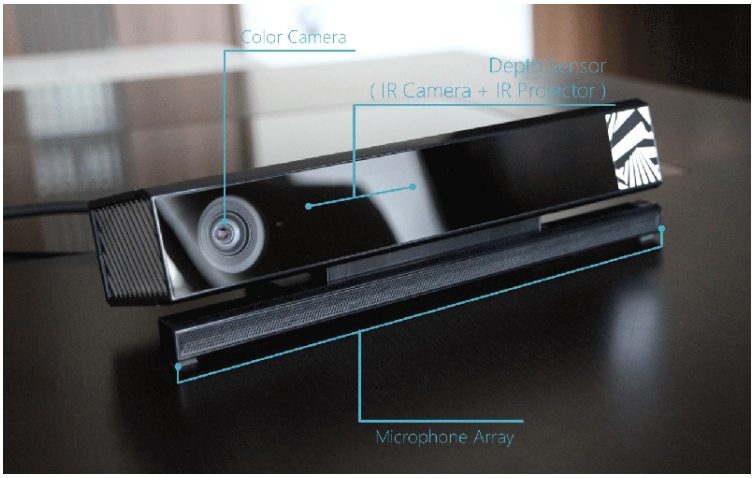

Figure 1. Microsoft Kinect v2.

1.2 Game Engine

A game engine is a software framework for creating and developing video games. Its core functionality includes rendering frames, collision detection (by means of a physics engine that implements the relevant laws of physics in the game), sound, scripting, animation, artificial intelligence, networking, memory management, threading and scene graphs. A game is built on this game engine by writing scripts and attaching to it. The script controls what each component does and when. Many game engines are available in the market and Unity game engine [4] is one of the most widely used. It supports the main features required for this project as shown in the Figure 2. It has also won many awards for being the best game engine. It is a cross-platform game engine developed by Unity Technologies. This software is compatible with Windows and Microsoft have released a Kinect package that is compatible with this game engine. The data collected from Kinect can be accessed using

the Kinect package for Unity. Player movements are converted to basic motion in the script. The main game logic is designed by writing such scripts and attaching it to the project file. Each script file has Start and Update functions, which are individually incorporated into each script file. Start functions are called at the beginning of the game and update functions are called at every frame. Levels are also designed using a text file that defines the play environment. Levels are developed at the beginning of the game by reading these text files.

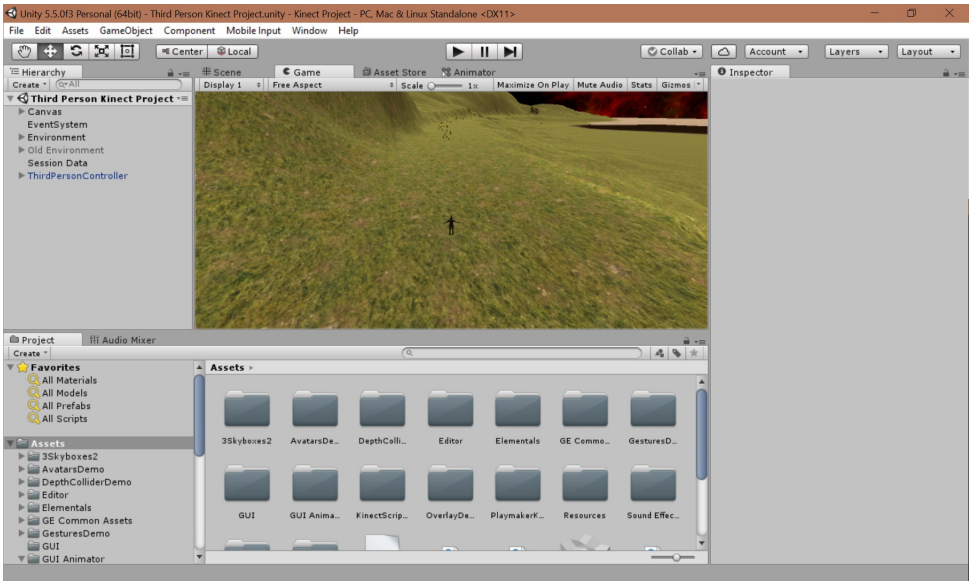

Figure 2. Snapshot of the Unity game engine editor

CHAPTER 2

SYSTEM DESIGN

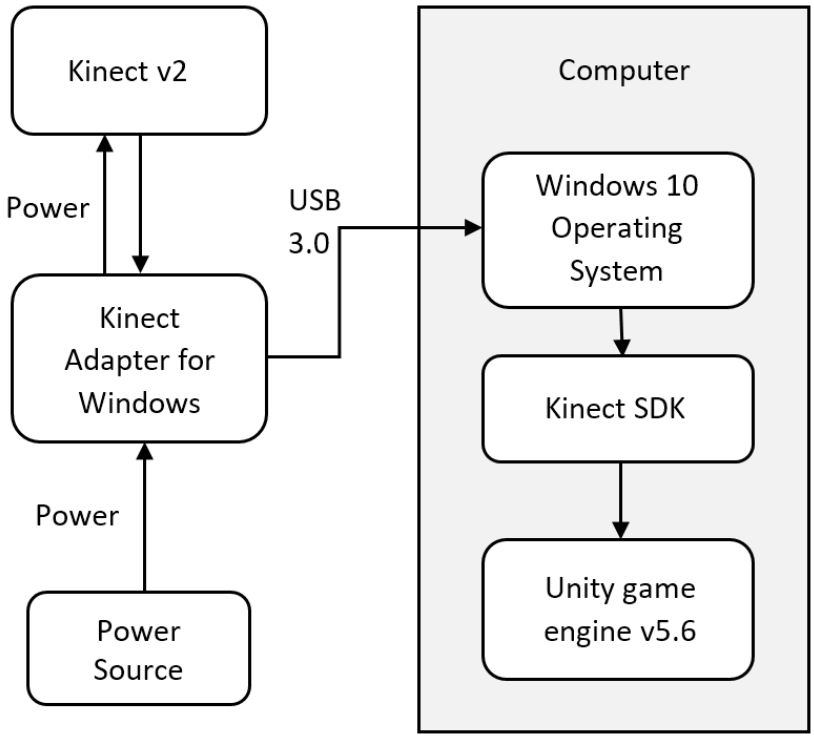

Figure 3. The System Design flow.

This chapter describes how the system, including hardware and software, is designed. As shown in the Figure 3, a Microsoft Kinect v2 device is connected to a computer. Because the Kinect v2 is originally designed to work with the Microsoft Xbox One game console, it has a port that is proprietary to Microsoft. To enable computer use, Microsoft also released an adapter that acts as a power source and USB 3.0 port converter. Therefore, the Kinect can be connected to the computer using the Kinect Adapter for Windows module. USB 3.0 should be used because of the high amount of data transfer from the Kinect to the computer. The Kinect must also be

connected to an external source through the Kinect Adapter for Windows module. Along with this adapter the drivers to access the full functionality of the Kinect were obtained by installing Kinect software development kit for windows (SDK), to allow Windows to communicate with Kinect. In addition to this SDK, Microsoft has also released a package to allow the Unity game engine to access the Kinect SDK. This package, Kinect for Windows Unity Pro package v2.0, was attached to the project in the unity editor. The flow of the Kinect input to the Unity game engine is shown in the Figure 4.

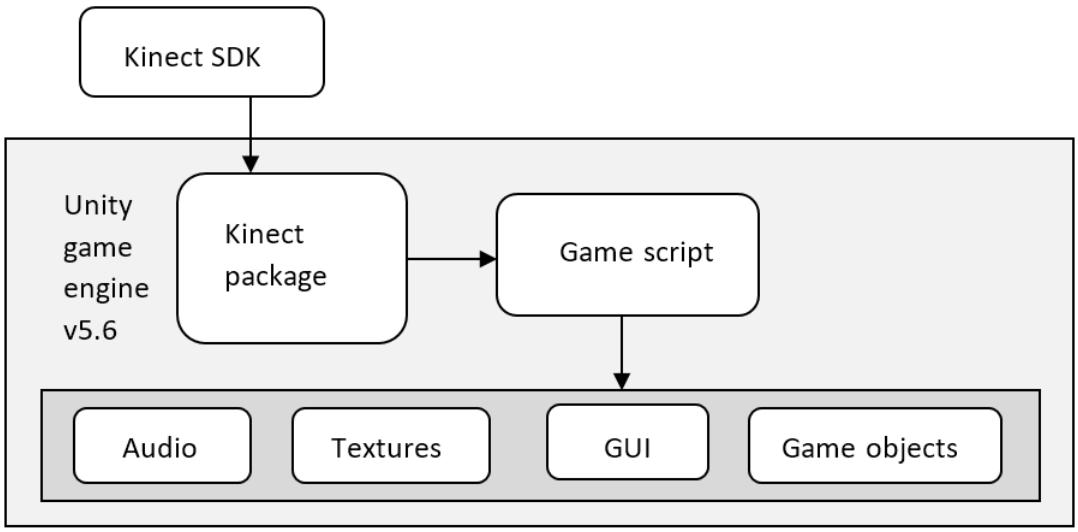

Figure 4. Kinect input flow chart for Unity game engine.

2.1 Movement capturing

The player (child) who is playing the game, jump, hop and duck in various sequences to win the game. The player must perform these actions in front of the Kinect device to capture the players movement. Kinect captures both RGB and depth images of the player. These captured images are sent to the computer via

USB 3.0. These images are processed by the Kinect drivers installed with the SDK. Kinect for Windows Unity Pro package v2.0 processes the data and gives the positions of the body nodes of the player, which are shown in the Figure 5.

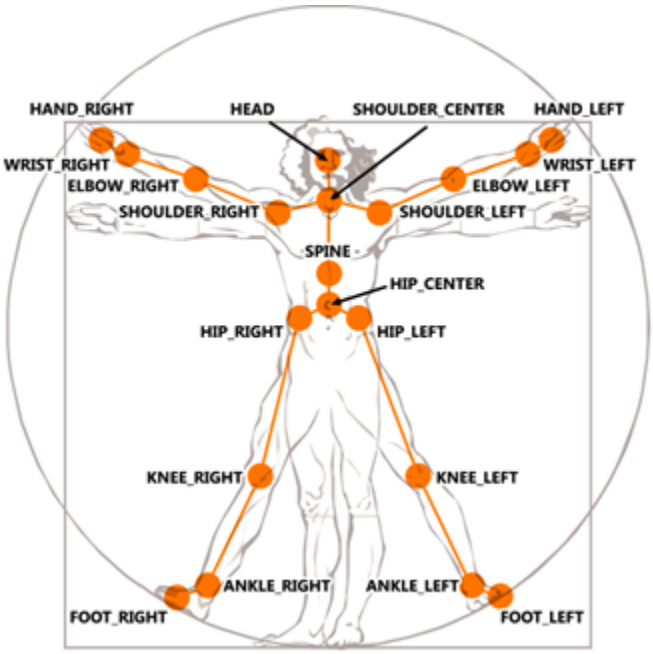

Figure 5. Body Joints detected with the help of Kinect input

These joints shown in the Figure 5 are accessed in the game script by calling the in-built Kinect for Windows Unity Pro package v2.0 functions.

The Kinect device can track up to six people's movements. Out of all these people, the game will take the data of the last tracked person. This data is collected for every frame. Out of all these joints, the game measures the position of the base of the spine (HIP_CENTER in the Figure 5) of the most confident person Kinect can track. Based on the position and movement of this joint, the game engine determines the motion of the in-game character (more information on how this movement is processed is described in Chapter 3).

2.2 Game Design

The design of the game mainly consists of menu screens, gameplay, UI elements, game objects, textures and Audio.

2.2.1 Menu screens

There are 10 screens: Main Menu, Finding Center (which locates the base of the player's spine) , Setting Difficulty, High Scores, Level Start, Pause Menu, Game Over, Game-play UI, Finish and Lost Life screens. Every screen has a Kinect Tracking indicator on the top center of the screen. A red square at the top of the screen indicates that the Kinect is not tracking the child. Green indicates that it is tracking.

1. The Main Menu has three buttons: Start, High Scores and Exit, along with an Audio Toggle option shown in the Figure 6. The Start button will take the user to the Finding Center screen, which uses the Kinect to determine the base of the players spine. Selecting High Scores will take the player to the High Scores screen. The Audio Toggle mutes and unmutes the audio. Every button has an event handler function, which is called to disable or enable UI screens and game objects. The Exit button quits the game. The logic and flow of the Main Menu is shown in the Figure 7.

Figure 6. A screen shot of the Main Menu Screen showing the title of the game 'Bhaago'.

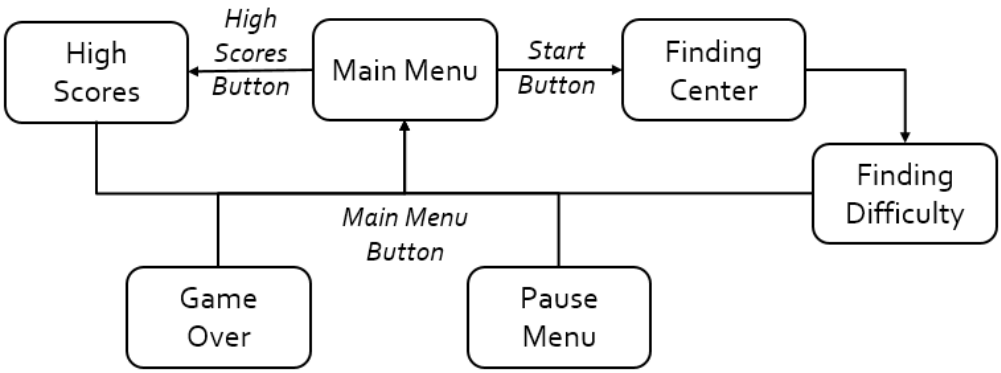

Figure 7. Logic and flow of the Main Menu.

2. The High Scores screen displays the top five scores and maximum height jumped by previous players, as stored in the game memory. An example of this is shown in the Figure 8. A Main Menu button returns the player to the Main Menu screen.

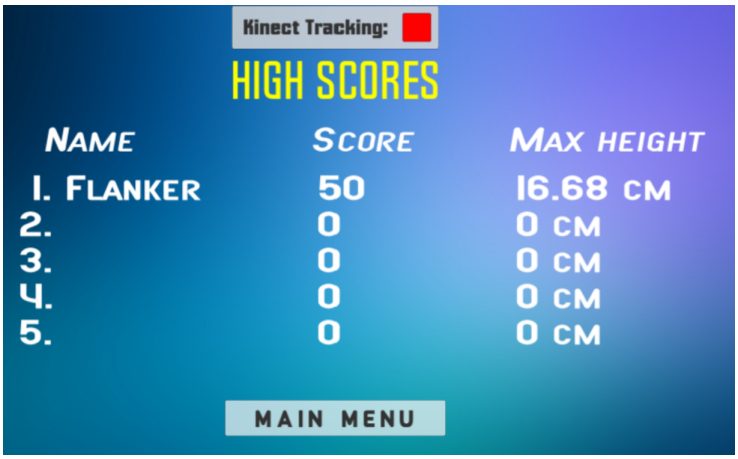

Figure 8. A screen shot of the High Scores Screen.

3. The Finding Center screen appears as soon as the player presses the Start button on the main menu screen, as shown in the Figure 9. This interface gives the player 3 seconds to calibrate their center position. It does so by receiving input from the Kinect for 3 seconds. The timer is paused when the Kinect cannot track the player. As soon as the 3-second period ends, the center position is calculated by averaging all the positions obtained during this time. Jump threshold and duck threshold are calculated from this calculated height position. These thresholds are used to identify if the player is jumping or ducking. The game then moves to the Setting Difficulty screen after all these calculations.

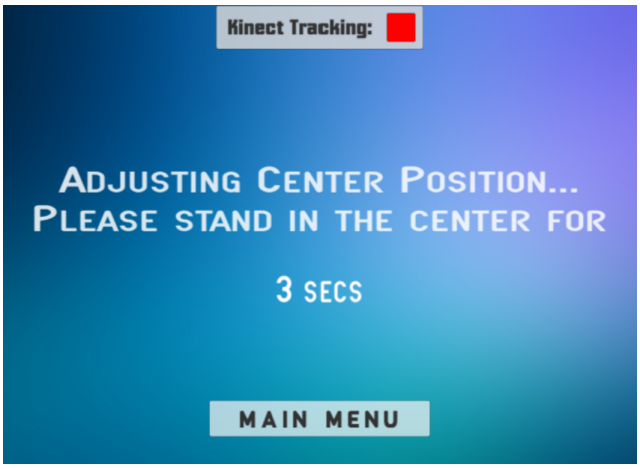

Figure 9. A screen shot of the Finding Center Screen, showing time remaining for the Kinect to track the player.

4. The Setting Difficulty screen asks the child to jump as high as possible, an example of this screen is shown in the Figure 10. From this obtained maximum jumped height, the difficulty is calculated. This value is used to manipulate the jump height, duck height and sensitivity required to move sideways. By this process, the dynamic difficulty feature is achieved, which adjusts the game inputs based on the amount of jump a child can perform. Pressing the Start button on this screen takes the player to the Level Start menu screen. The player must jump a minimum of 10 cm to start the game.

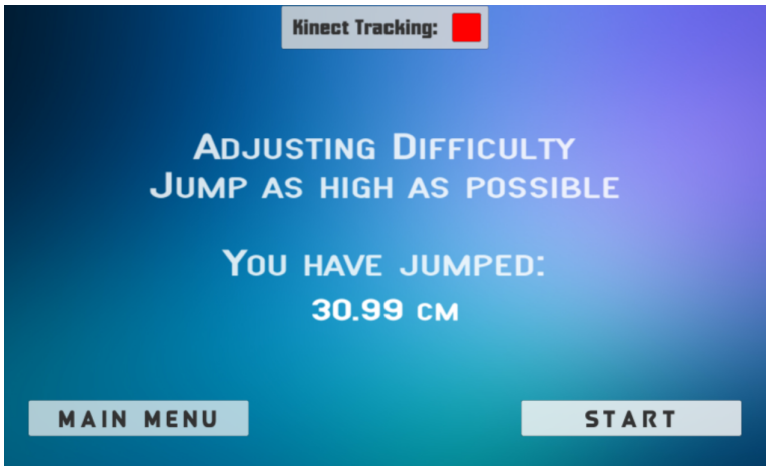

Figure 10. A screen shot of the Setting Difficulty Screen, showing the maximum distance jumped by the child.

5. The Level Start screen appears before a level starts. Its duration is 2 seconds, as shown in the Figure 11. It notifies the player that the level is about to start. As soon as the player presses the Start button or Restart button and when the in-game character touches the finish line (except on the last level), the first level, same level or the next level is started respectively. This screen lasts for 2 seconds with a countdown sound, which indicates that the player has to get ready to play the level. The game play starts after this screen times out.

Figure 11. A screen shot of the Level Start Screen, before level 2.

6. The Game-play UI screen (In-game UI screen) has UI elements. They include three boxes (sections). The first box from the left displays player's score, health bar to show health remaining, lives and invincibility power-up bar to show the amount of invincibility left (more about each of these in Chapter 3). The second box shows the current level being played. The third box on the right end, displays the maximum height jumped by the player in that game session. All these elements are displayed over the game-play, as shown in the Figure 12. The flow of the game-play UI screen is shown in the Figure 13.

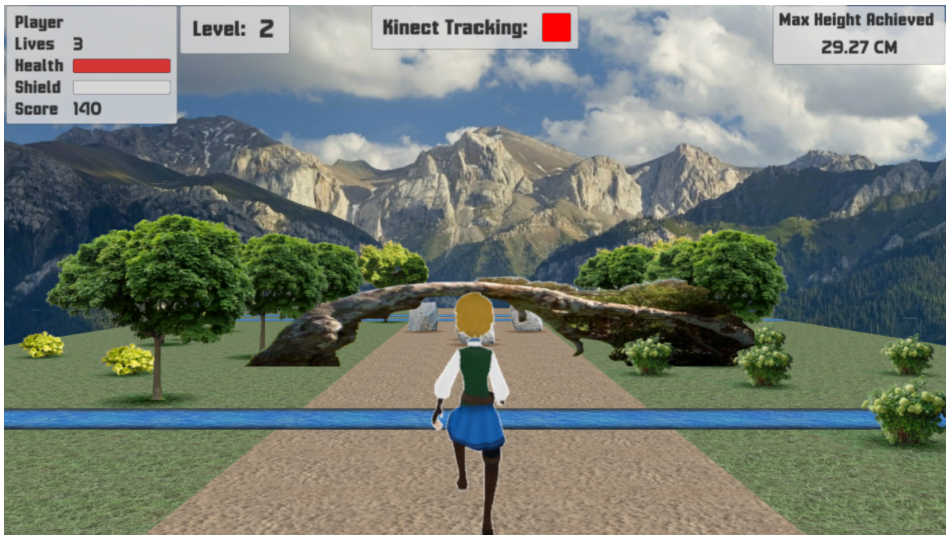

Figure 12. A screen shot of the game-play, showing UI elements.

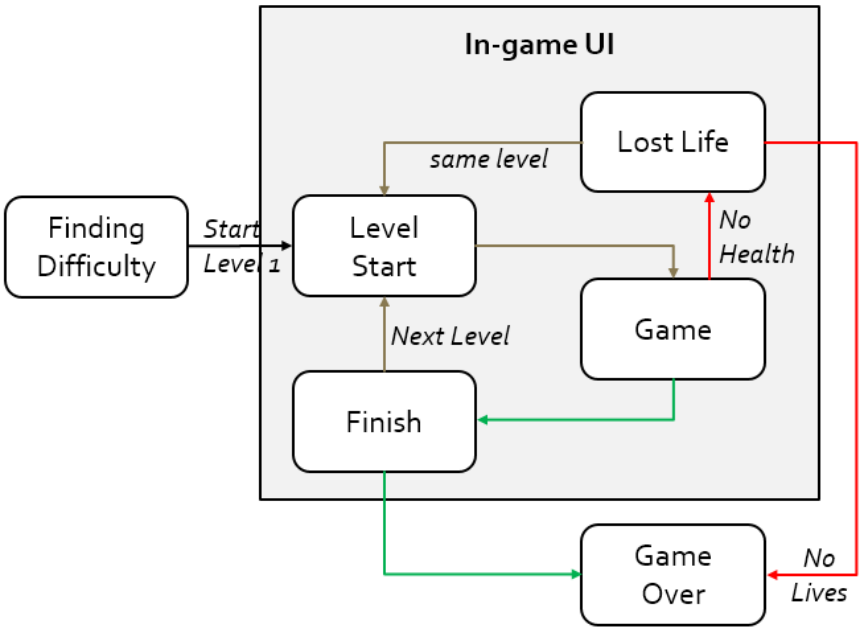

Figure 13. The flow of the Game-play UI screen.

7. The Pause Menu appears as soon as the player presses the Escape button on the keyboard, as shown in the Figure 14. This menu has three options: Resume,

which continues the game; Restart, which resets the game on Level 1, (that is, the same function as Start on the Main Menu); and Main Menu, which returns the user to the Main menu. The UI flow for this screen is shown in the Figure 15.

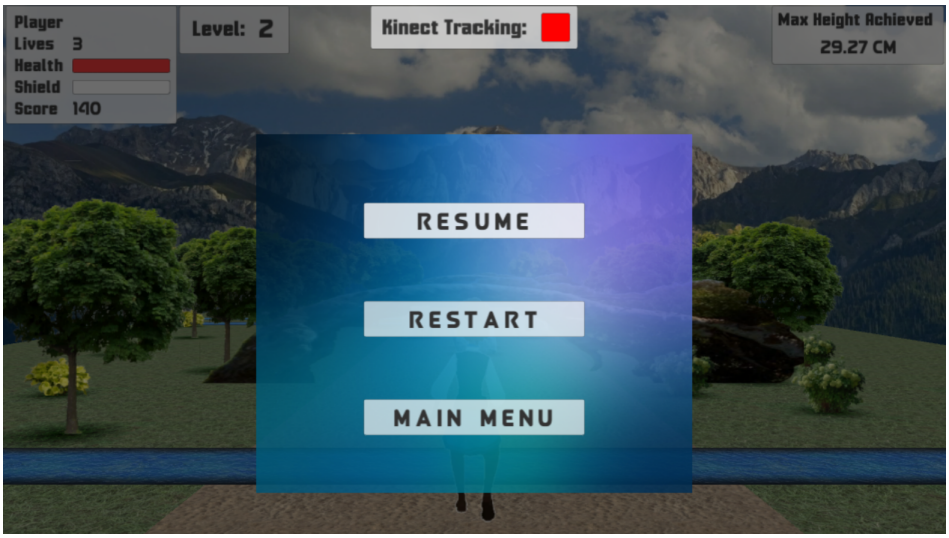

Figure 14. A screen shot of the Pause Menu screen.

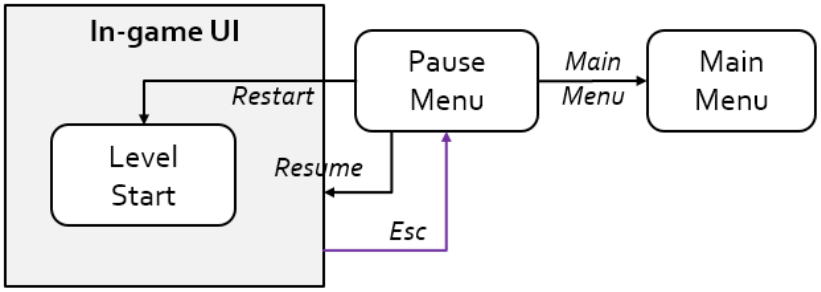

Figure 15. The flow of the Pause Menu screen.

8. The Lost Life screen displays nothing. But, it is used to pause the game for 1 second. It helps the child to show that the player lost its life. The number of remaining lives drops by one when the players health drops below zero. If

these lives drops to zero, then the game is over. The Lost Life screen is shown for 1 second and then the level restarts. The flow of this screen is shown in the Figure 13.

9. Like the Lost Life screen, the Finish screen also doesn't display anything. It appears as soon as the player arrives at the finish line on a level, as shown in the Figure 16. The flow of this screen is shown in Figures 13 and 19. The Figure 13 shows the flow when the level is not the last level and the Figure 19 when the level is the last level.

Figure 16. A screen shot of the player reaching the Finish line.

10. The Game Over screen appears as soon as the player reaches the finish line on the final level, which means the player has won (Figure 17) or when the player loses all his or her lives, which means the player has lost (Figure 18). It shows the score obtained and a Main Menu button to allow the player to return to the Main Menu. The flow of this screen is shown in the Figure 19. If the player wins the game and the score is one of the five highest scores, a box appears

with a message to input the player's name. This name is then displayed on the High Scores screen in descending order of the score obtained. The previously fifth-highest score is discarded.

Figure 17. A screen shot of the Game Over screen, when the player wins and achieves a High score.

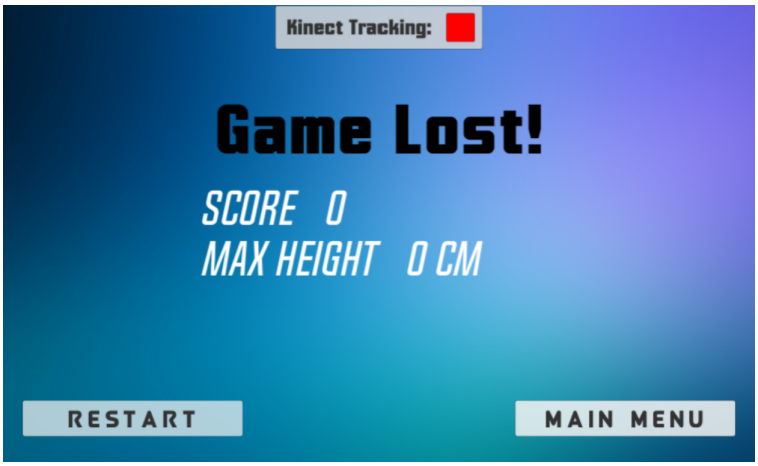

Figure 18. A screen shot of the Game Over screen, when the player loses.

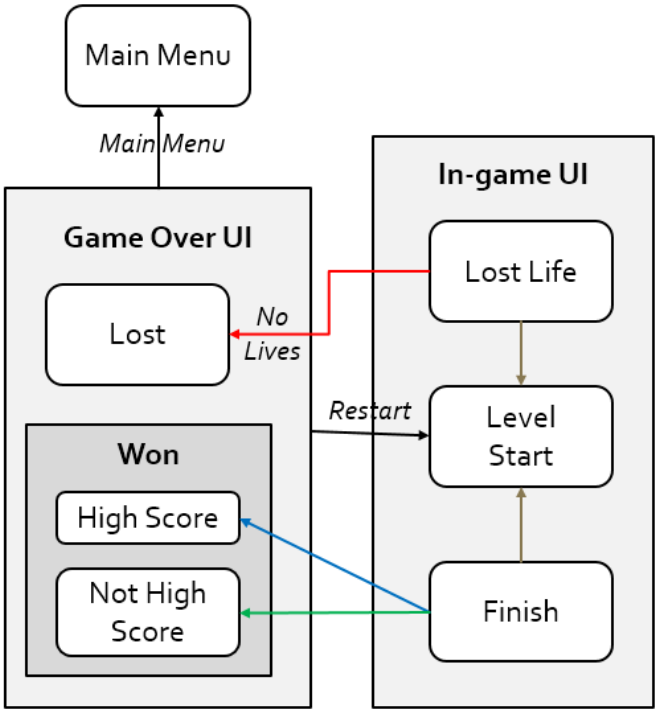

Figure 19. The flow of the Game Over Screen.

2.3 Game-play Design

The in-game character appears at a distance in front of the camera as shown in the Figure 12. The ground is divided into tiles (rows and columns) as denoted in a level design file, which is a text file input to the game engine. Each character in the level design file describes the type of each tile. Each level has 15 columns, with middle three columns as the path containing obstacles and the remaining tiles displaying grass and other environmental objects.

By default, the character continuously runs forward. The player must jump, hop, duck or move sideways to avoid the obstacles. The background scenery remains distant, although it does appear to come slightly closer.

2.3.1 Game Objects

This game has 11 game objects: Player (in-game character), Back camera, Tile, Scenery, Stones, Logs, Trees, Bushes, Crocodiles, Health power-up and Invincibility power-up. This game is rendered in 2.5D, meaning that all these objects are planes in a 3D space with different sizes and respective sprites. All the objects have a pivot point at their respective centers. This game has Y axis running vertically, Z axis pointing to the front and X axis running horizontally.

1. **Player**: This is the games protagonist. This object is moved with respect to the inputs received by the game. It has three sprite images with UV animations for running, jumping and ducking. The height of the player is 5 units above the ground (which has a value of zero) and a width of 2.5 units. The running and ducking animations are created by rotating through 12 sprites. Samples of these images are shown in Figures 20, 21 and 22.

Figure 20. A sprite of Player running animation.

Figure 21. A sprite of Player jumping animation.

Figure 22. A sprite image of Player ducking and running (simultaneously) animation.

2. **Back Camera**: This is the default game view. It follows the player at a distance of 20 units on the Z axis and is raised by 5 units on the Y axis.

3. **Tile**: Tiles are used as the ground. This game object is based on the level design file. There are ten tile types: path, lava, water, river, stone, log, tree, grass, bush and finish. Each type has its own unique texture, as shown in the Figure 23. Game objects are also assigned to each item based on its type. These assigned game objects take the tile's position. Grass, bush and tree types are used for the environment and appear on the sides of the path. River, water, pond, stone and log are used for obstacles. Finish is used to end the level.

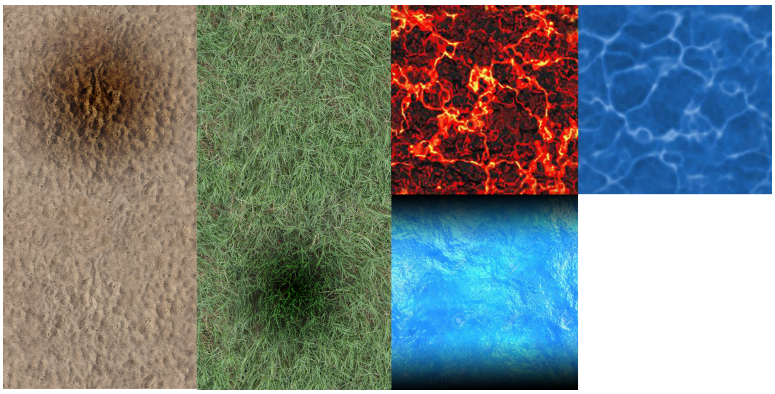

Figure 23. Different types of textures for tiles.

4. Scenery: This object remains in front of the player and increases slowly in size as the player progresses to create a parallax effect. It depicts mountains, as shown in the Figure 24.

Figure 24. Example of background scenery used in the game.

5. **Stone**: This game object is assigned to tiles that are designated as stone type in the level design file. The stone appears in the center of the tile, facing the player on the Z axis. It has a height of 2.5 units. The player must avoid this tile

by jumping over it. Colliding with it will lower the player's health. Textures are chosen randomly between the two options shown in the Figure 25.

Figure 25. Example of the textures of the stone object.

6. **Log**: This has the same properties as stone, but appears with a gap underneath. The player must duck under it to avoid the collision. Logs take up the entire row of the path (3 tiles wide). The log game item appears with a gap under which the player can pass, as shown in the Figure 26.

Figure 26. Texture of the log object, a bent tree with gap to pass under it.

7. **Tree and Bush**: These are used to decorate the game and are present on the grass, but not on the players path. Textures are randomly chosen among these, shown in the Figure 27.

Figure 27. Examples of textures of the objects bushes and trees used in the game.

8. **Health and Invincibility**: These are power-ups. Health replenishes the players health to full and invincibility makes the player invulnerable to any collision for some distance. Icons for these power-ups are shown in the Figure 28.

Figure 28. Icons of the Health and Invincible power-ups used in the game.

2.3.2 Audio

This game has 11 audio files and 5 audio sources. As the unity game engine can play only one audio file on an audio source at a given time, there is a need to

use multiple audio sources to give a better feedback for each action. The first audio source is only used for the player running sound, which is played on a loop. The remaining audio sources cover additional sounds as listed below. If an audio source is busy, then the next one is chosen to play. To avoid conflicts, more than one audio source is chosen. The sounds are as follows.

1. Player running (audio loop) while the player runs
2. Player jump start when the player jumps
3. Player landing sound when the player lands on the ground after jumping
4. Player duck sound when the player ducks
5. Player hurt when the player collides with an obstacle
6. Level start as soon as the level start screen appears
7. Finish as soon as the player reaches the finish tile
8. Power-up when the player takes a power-up
9. UI-Button for button clicks
10. Game Over - Win: when the Game Over - Win screen appears
11. Game Over - Lost: when player loses all his or her lives and Game Over - Lost screen appears

CHAPTER 3

IMPLEMENTATION

The game is developed using a C# script file with instructions to change screens, change height parameter, change audio volume, load levels, start and end the game, move objects, calculate collisions and save high scores. The Figure 29 shows the basic structure of a Unity C# script file. It has two main functions: start and update. The start function in every script file is called at the start of the game. The update function is called for every frame. Attaching the script file to an object indicates the object's script file.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Example : MonoBehaviour {
6
7       // Use this for initialization
8       void Start () {
9
10      }
11
12      // Update is called once per frame
13      void Update () {
14
15      }
16  }
```

Figure 29. Structure of a Unity script file, with start and update functions.

3.1 Level Design

As mentioned above, level design files contain information on the position and type of every tile on the map. It is designed to be as simple as possible so that it

can be easily understood and modified by a non-programmer.

An example of a few lines of level design is shown in the Figure 30.

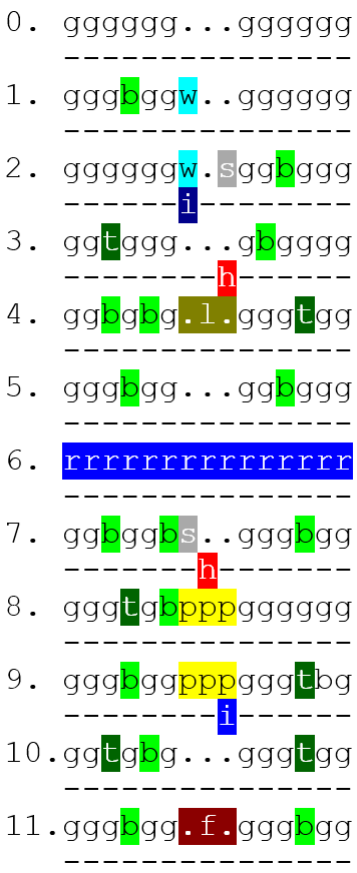

Figure 30. Example of a Level design file.

In the Figure 30, we can see that there are 24 lines of characters. Every two lines of characters gives information for one row of tiles. Two lines were used in the code for every row in the game so that power-ups could be implemented without conflicting with the main tile type. For example, a power-up can appear on a river tile. The first line in each pair of lines indicates the tile type, that is, path, river, water, lava, tree, bush, stone, finish and log. These are indicated by ., r, w, p, t, b, s, f and l respectively. The second line indicates whether there is a power-up and if

so, whether it is Health or Invincibility, which are denoted by h and i, respectively. Therefore, in the example given in the Figure 30, we have 12 rows and 15 columns of tiles. Details about the tiles are below. Apart from these notations, we have 3 other notations for randomizing objects. '0' denotes either a tree or a bush, chosen in random. Similarly, '1' for stone or water. '2' for log or river in that entire row.

Path Tile: For easy understanding, these tiles are denoted by "." in the level design file. This is the safe tile for the player to run.

River Tile: The river tile has the only animated texture, which mimics a river flowing. The time taken for the texture to completely move across the tile is set to 1 second; this means that the river flows at a speed of 5 units per second as the size of a tile is set to 5 units, as described below. The time difference between the frames is taken and is multiplied by this speed of 5 units per second; this gives us a constant speed even if the time between frames is inconsistent.

Water & Lava Tiles: Water tiles are used to indicate a single tile with no other water tiles around them. Lava tiles take up 6 tile spaces, covering all the path. Running over these tiles is taken as a collision by the player. Static maps are used on these textures for a more realistic feel of water.

Stone Tile: When a tile is set as stone, another plane with a stone image is placed at the center of the tile. The player must jump over these tiles to pass them or else it is counted as a collision. Either of the two stone textures described above are randomly selected to create a more diverse effect.

Log Tile: A log tile covers the entire path. So, it is only denoted in the middle row. No other types of tiles can appear in this row. The player must duck under it to avoid collision.

Finish tile: This tile indicates finish line. As soon as the player touches this tile,

the level ends and the next level starts. If the player makes it to the finish tile on the final level, then the game is won.

Bush and Tree Tiles: These tiles are used for environmental objects. There are two types of trees and bushes which are randomly selected. These tiles are not to be placed on the path.

Health and Invincible Power-Ups: Health power-ups replenish the players health to the maximum level. Invincibility power-ups make the player invulnerable to any collisions for a certain distance. The power-up objects are disabled as soon as the in-game character collides with them and takes the power-up.

Positions: The size of each tile measures 5 5 units. The central tile in each row is centered on the X axis. All tiles immediately border their neighbors. As the size of the tiles is equal and there is no gap between tile edge, all the tiles combined create a single plane. Therefore, it was vital to choose textures with seamless borders, so that they visually connect with adjacent tiles.

Levels: Level text files are denoted by the name level followed by an integer that shows the level number. Level numbers start from 1. For example, the level 3 text file is called level3.txt. These level text files are placed in the Assets/Levels folder in the main game folder before building the game.

Note: Hopping is achieved by placing river tiles alternating with the path for 2 or 3 consecutive rows. To cross this, the in-game character must jump to cross the river, land on the path and immediately jump again to cross the next river.

The critical issue with this kind of game is when children repeatedly play the game, they may become disinterested because of the repetitiveness of the game. To overcome this problem, the game should be coded so that level design is not repeated during successive runs. To allow non-repetitive levels, we implemented random ob-

stacle generation. For example, stone and water game items (which cover one tile each) and log and river items (which cover a whole row on the path) should appear randomly. The letter l or r placed on the center column of a row replaces that row with a log or a river by modifying that entire row. The entire row will be replaced with gggggg.l.gggggg or rrrrrrrrrrrrrr in the games memory.

Additionally, to make the game more interesting, different packages with different of textures exist in the game. These textures and objects, described in section 2.3.1 , belong to package A. Similarly, two more packages, which have different environmental textures, like desert and snow, are available. Each package has a texture for each object and the type of the tile. These packages are randomly applied to the levels to maintain player interest.

3.2 Input: Kinect

This game uses Kinect to monitor the players position. However, a mouse and keyboard are needed to press the initial menu buttons and enter the name in the High Score screen, respectively. As mentioned earlier, Kinect is used to locate the players Spinal Base (Hip Center), which is the base of the spinal cord. Kinect input is taken only when the child is a distance of 110 cm away from the device. This is done to allow the Kinect to receive more accurate data about player movements. Standing too close affected the input and gave erratic values. In such cases, the visual indicator at the top of the screen indicates to the player that the Kinect device is not tracking.

Calibration: As soon as the player presses the Start button, the Kinect begins to measure the position of the base of the players spine for every frame. When the screen is finding the center/start position, the data collected for every frame are

stored until the 3-second timer ends and the game starts. This timer pauses if the Kinect cannot track the player. The data are then averaged to determine the players starting position (center position). The jump and duck thresholds are adjusted from this position.

Actual Input: Once the game has found the start position, the game UI moves to the Setting Difficulty screen. From here, the Kinect input is processed to generate smoother output for every frame. This reduces the jitter and any irregularities in the input. The game stores the input position in a stack of three elements , discarding the then fourth-and-last element. These three positions are then averaged to determine the Kinect input. This averaged input is used in the game. These averaged data are again stored in another stack of three elements with the applicable time-stamp for that instant, as shown in the Figure 31. This way, the game keeps track of the last three input positions and their time stamps, which calculates the players movements.

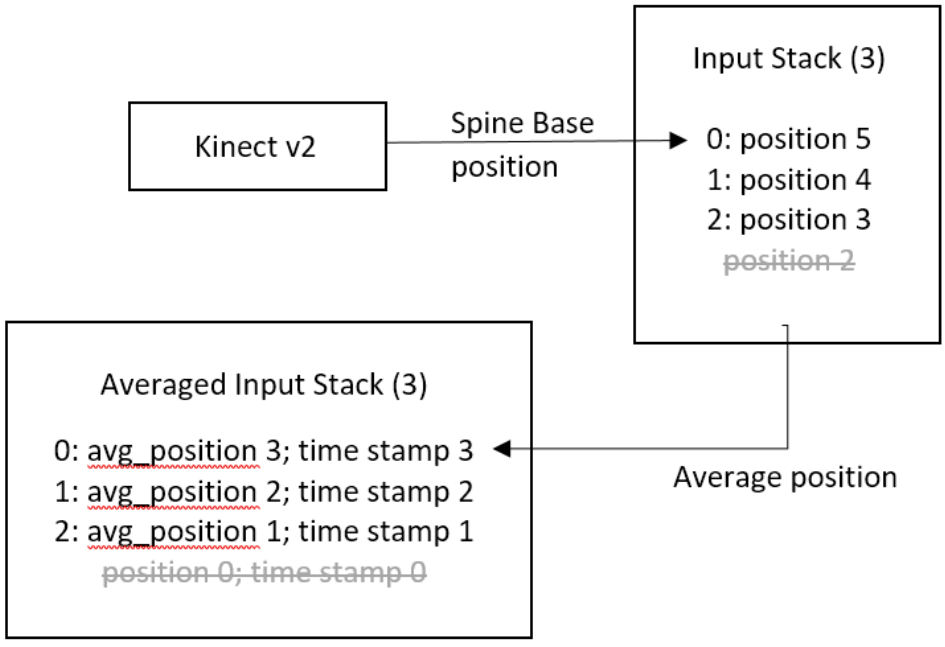

Figure 31. Flow of a Input position.

On the Setting Difficulty screen, the player is requested to jump as much as possible. A minimum height of 10 cm is required to start the game. This difficulty factor affects the jump speed, duck distance and sideways movement. This game is designed by keeping the maximum height reached by the child as 30 cm, for which the game character can jump at a maximum speed of 25 units per sec. So, to determine the difficulty level, this jump height (in cm) is divided by 30 to give a difficulty factor of 1 here, which is a multiplicative identity. This difficulty factor is adjusted to the present child's height by dividing with the designed height of 30 cm. If the difficulty increases, then the child has to jump higher for the same result in the game.

3.3 Player Movements

Camera & Scenery: As described previously, the back camera is placed behind the player and faces in the positive Z axis direction. It follows the player only along the Z axis. Such placement of the camera creates a third person perspective; therefore the game is defined as a 2.5D game rather than a 2D game. Every plane, except the tiles in this game, face the negative Z axis because the back camera, which faces the positive Z axis, is the default view. As every object faces the camera, object textures can be seen. In addition to the tiles and game items, a large plane with a mountain backdrop texture is placed in front of the in-game character. This plane also moves as the player progresses along the Z axis. This creates a parallax effect that gives the illusion that there are mountains in the distance.

Run: By default, the in-game characters speed is set to running. For every frame, the in-game character moves a set distance along the positive Z axis, away from the origin. This is the pace of the game.

Jump: For the in-game character to jump, the player must exceed a jump threshold,

which is calculated as soon as the start position is calculated. This value is set to 0 cm on the positive Y axis by default. These threshold values are reduced by reducing the Kinect ratio slider in the options menu. The jump threshold is calculated by first multiplying the default value by the Kinect ratio and then adding the value to the calculated start position. The game checks the latest averaged input position with the thresholds. If the player exceeds the jump threshold, then the speed of the jump is calculated by dividing the distance between the latest and the last stored position with the time difference. This time difference is calculated using the time stamps of those two positions. This speed obtained is now converted to the in-game worlds units along the Y axis. With this speed, the in-game character is translated along the positive Y axis (upwards) every frame. This jump speed is reduced in each frame by the gravity factor and eventually becomes a negative value, at which point the in-game character begins to fall. The in-game character's jump ends when the player reaches the ground and the player state is set to running again. The difficulty factor is applied to the speed by dividing the speed with difficulty factor. If the difficulty factor is high, then the speed will be less than normal. To convert Kinect units into in-game character's confined space in the game, Kinect input units are multiplied by 0.12 for jumping.

Duck: Similar to jumping, the averaged input position is checked with the duck threshold, which is by default set to 11 cm below the starting position. If the player goes below this threshold, the in-game character ducks. In this state, the distance from this averaged input position and the duck threshold is calculated and converted to the in-game units. This obtained distance moves the in-game character along the negative Y axis. Ducking ends when the player exceeds the duck threshold and the state of the in-game character is set to running again. Similar to jumping, the

distance moved by the player is divided by the difficulty factor , so the in-game character moves less than normal for the same movement on a higher difficulty setting. To convert Kinect units to game units, the Kinect input units are multiplied by 0.15 for ducking, similar to jumping.

Side movements: Side movements of the player are calculated for every frame. The distance between the averaged input position and the starting position is calculated along X axis and converted to in-game units. This distance is used to move the in-game character to that position along the X axis (negative X axis for the left side and positive X axis for the right side). Similar to ducking, here the difficulty factor determines the size of the in-game character's side movements. To convert Kinect units to the game units, Kinect input units are multiplied by 0.125 for sideways.

The above inputs allow the player to move the in-game character in all axes simultaneously. In-game character movements are restricted to the game-play area. This prevents any erratic inputs from the Kinect device.

3.4 Other objects

To improve game efficiency, only 30 rows of tiles are instantiated at the start of the game. Similarly, only 15 objects of each type are instantiated at the start. Tiles that are programmed to have objects in their positions choose from the unassigned instantiated objects depending on the object type. As described above, power-ups are treated differently in the code to other objects to allow a tile to have both an object and a power-up. As the player progresses, the tiles and objects that are behind the in-game character at a distance greater than one tile are unassigned and reset in the furthest row from the character with the next row from the level design text file. In this way, the efficiency is improved by always showing a limited number of

objects on screen. When a new row exceeds the number of lines in the level design text file (that is, rows that appear beyond the finish line) , then the game generates tiles beginning again from the first row. This creates the illusion of objects even beyond the finish line.

	0	1	2	3	4	5	6	7	8	9	10	11	0	1	2	3
1	0	1	2	3	4											
2		1	2	3	4	0										
3			2	3	4	0	1									
4				3	4	0	1	2								
5					4	0	1	2	3							
6						0	1	2	3	4						
7							1	2	3	4	0					
8								2	3	4	0	1				
9									3	4	0	1	2			
10										4	0	1	2	3		
11											0	1	2	3	4	
12												1	2	3	4	0

Figure 32. An illustration of how tiles are recycled.

the Figure 32 is an example, with 12 tiles of rows in a level and 5 visible rows. The left most column shows the number of tile rows player has passed. The top row indicates the row number of the tiles. Notice that when player passes first row of tiles, then the tiles of 0th row will take the properties and the position of the 5th row. Respective objects which were associated to these tiles are deactivated and new objects, which are inactive are activated and associated. In this way, even the objects used along with the tiles are recycled. This reduces burden on the game engine, making the game more efficient.

3.5 Collision

Collision is only checked between the in-game character and the objects. As tiles that are behind the in-game character are recycled, the in-game character appears between the first and second rows of tiles in any given frame. Moreover, the player path comprises only the middle three rows. This reduces the number of calculations the game must perform to only the six tiles that surround the in-game character.
Run: When the in-game character is in the running state, the character is always in contact with the ground and the distance between water, pond, river, stone and log tiles are checked. If the distance along the Z axis is below 40% of the size of the tile (that is, 2 units) or if the distance along the X axis is less than 40% of the sum of tile size and the in-game characters width when the character is alongside an object, then a collision is recorded.
Jump: While jumping, the in-game character is not in contact with the ground, so only the stone and log tiles are checked. Additional to the Z and X axes in the running state, the Y axis is also checked by adding half the in-game characters and objects heights to the distance to the tile. the Figure 33 shows how these constraints are checked in a 3D space, even though the player and the stone are planes.

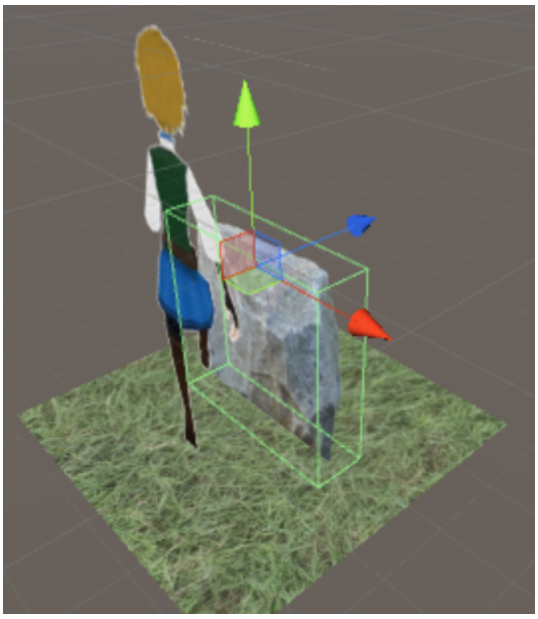

Figure 33. An example of an imaginary collision rectangle over a stone (plane).

Duck: Collisions for water, pond, river, stone tiles are checked as in the running state. However, for the log, height is also checked. If the in-game characters height from the ground tile is more than 30% of the player's original height, then a collision is recorded.

Finish line and Power-Ups: Collisions are checked along the Z axis regardless of the players state. The player can finish or obtain power-ups by entering that tile. If the in-game character touches the finish line, then the game proceeds to the next level. If the player finishes the last level, then the game is won.

Hit: If the player is Hit, then the players health is reduced by 25%. If the player loses complete health, then the player loses a life. After losing three lives, the game is Lost. For every life lost, the current level restarts.

3.6 Score

The score is calculated every time the in-game character passes a row of tiles and the tiles are reassigned. If these tiles contained obstacles that the player successfully avoided, then a score of 10 points is awarded. However, if the in-game character collides with an obstacle, 10 points are immediately deducted. When a level restarts because the player lost a life, then the initial score when the player previously started that level is restored.

When the in-game character reaches the final tile of the last level, his or her score is compared with the stored high score data. If this score exceeds any of them, then the score appears in the highest eligible place and the player is asked to enter his or her name on the Game Over - Win screen. When the player enters his or her name and presses the Main Menu button, the score, player name and maximum height achieved in that session are stored. The previously-fifth-placed score is discarded.

CHAPTER 4

CONCLUSION

In this project, a 2.5D scrolling game using a Kinect device was built for children with intellectual disability. This project is intended to help the Kinesiology department at Texas A&M University Corpus Christi. It will enable children to overcome any access difficulties and practice. This will help them with knee stiffness and to improve jumping performance. It will also help in improving basic locomotive skills. These efforts help them build confidence and improve sporting performance and improves participation safety. The available existing games that use Kinect do not serve the specific needs of children with intellectual disability and are only available for Xbox. These limitations served as motivation to develop a Kinect-based game for Windows to help children with intellectual disability. We implemented a dynamic difficulty adjustment feature at the beginning of the game to customize the difficulty level according to the players ability. The game adapts to the maximum height the child can jump, which is crucial for children who are not as able to use these locomotive skills. Experimental results reveal that the dynamic difficulty adjustment is an efficient method to control difficulty compared with calculating this based on player height or a slider to adjust difficulty (which was initially planned). Moreover, a customizable level design feature was also implemented; this allows the trainer to edit the level files according to their preferences using Unity editor. The ability to read files from external sources was not included because of problems caused by errors with unknown alphabets. This game could therefore be further developed to accommodate this feature and handle such errors. However, the initial objective of allowing tutors to customize levels for their children was achieved.

The critical issue with games of this kind is that when children repeatedly play the game, they tend to become disinterested due to the repetitiveness of the game. To overcome this problem, the game should be equipped with non-repetitive levels that present different obstacles during successive runs. To enable non-repetitive levels, we implemented random generation of similar obstacles before each level starts.

Other than this, to make the game more interesting, different game packages are available. All the textures and objects described above belong to package A. Two additional packages can be implemented to create different environments like the desert and snow. Each package has a different texture for each object and type of tile. These packages can be applied to the levels at random.

The main advantage of this game is that it helps children to practice their skills even when they are unable to go to their nearest activity center for any reason. This may include poor transportation, distance from the center, parents busyness or adverse weather conditions. Practicing their skills will help them to adequately develop the loco-motor skills that form the foundation to acquire sport-specific skills and engage in physical activities their entire lifespan, such as jumping, hopping, ducking and running.

CHAPTER 5

FUTURE SCOPE

The current design is based on the conceptualization of building a game specifically for children with intellectual disability. The following points should be included in the future enhancement of the existing system.

While the system is intended to help children with intellectual disability to practice locomotor skills, because of the unavailability of suitable children, we could not actually test the game on those children. We recommend that experiments be performed with the target audience (children with intellectual disability) to analyze the efficacy of this system.

Further, the main intention of this project is to develop a prototype; design aspects of the game are beyond the scope of the current implementation. To make the game more appealing to children by using fancy game objects, we recommend consulting with game designers. The current implementation is highly flexible and allows easy changes to game objects.

The current implementation does not facilitate dynamic generation of levels but a fixed set of three levels. To make the game more interesting to the children and encourage them to play for a longer time, we recommend implementing different asset packages for different levels. If this is done at random, it would add another element of interest for the children.

To make the game more unpredictable and interesting, the levels could also be generated randomly. A set of rules can be written to randomly place objects based on their type and specify a distance between objects. Based on the difficulty, the distance between obstacles and number of power-ups in a level could be reduced. As

the player completes levels, the difficulty will increase.

BIBLIOGRAPHY AND REFERENCES

[1] Domire, Z., and Challis, J. Maximum height and minimum time vertical jumping. Tech. rep., Journal of Biomechanics, 2015.

[2] Einarsson, I., Olafsson, A., Hinriksdottir, G., Johannsson, E., Daly, D., and Arngrimsson, S. Differences in physical activity among youth with and without intellectual disability. Research report, Med Sci Sports Exerc., 2015.

[3] Hassani, A., Kotzamanidou, M., Tsimaras, V., Lazaridis, S., Kotzamanidis, C., and Patikas, D. Differences in counter-movement jump between boys with and without intellectual disability. Tech. rep., Res Dev Disabil., 2014.

[4] John, H., and Brian, M. A history of the unity game engine. Tech. rep., Worchester Polytechnic Institute, Bali, indonesia, 2016.

[5] Mcerlain-Naylor, S., King, M., and Pain, M. Determinants of countermovement jump performance: a kinetic and kinematic analysis. Tech. rep., Journal of Sports Sciences, 2014.

[6] Ryan, W., Harrison, A., and Hayes, K. Functional data analysis of knee joint kinematics in the vertical jump. sports biomechanics. Tech. rep., 2006.

[7] Schijndel-Speet, M., Evenhuis, H., Wijck, R., Empelen, P., and Echteld, M. Facilitators and barriers to physical activity as perceived by older adults with intellectual disability. Tech. rep., Intellectual and Developmental Disabilities, 2014.

[8] Stodden, D., Gao, Z., Goodway, J., and Langendorfer, S. Dynamic relationships between motor skill competence and health-related fitness in youth. Tech. rep., Pediatric Exercise Science, 2014.

[9] Stodden, D., Goodway, J., Langendorfer, S., Roberton, M., Rudisill, M., and Garcia, C. A developmental perspective on the role of motor skill competence in physical activity: An emergent relationship. Tech. rep., QUEST, 2008.

[10] Tanaka, Y., and Hirakawa, M. Efficient strength training in a virtual world. Tech. rep., IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), Nantou, Taiwan, 2016.

[11] Westendorp, M., Houwen, S., Hartman, E., and Visscher, C. Are gross motor skills and sports participation related in children with intellectual disabilities? Research report, Res Dev Disabil., 2011.