Gesture Based URI Development

Name of Game: Viking Run

Game Type: 3D Endless Runner

Gesture Technology used: Microsoft Kinect V2, Voice control

Group Members: Aaron Hannon (G00347352), Matthew Sloyan (G00348036), Michael Coleman (G00347650)

Github Link: https://github.com/aaronhannon/GestureBasesUIProject

Videos:

Project Demo: https://www.youtube.com/watch?v=9mKLsSUDwb4&feature=youtu.be

GitHub Repository Overview: https://www.youtube.com/watch?v=Ha1AnRz1Lms

Note: Test Plan with cases and class diagrams are available in github repository.

Outline of work:

Aaron Hannon

- Bug fixes throughout development.
- Kinect implementation and demo video.
- Day/Night cycle with lighting
- Random Chunk Generation
- Chunk Designs using Cinema 4D
- Pause Menu
- Boat Implementation
- Helmet Power-up
- Render Distance - Fog
- Heart/Life System

Michael Coleman

- Bug fixes throughout development.
- Project walkthrough video.
- Test Plan
- Coin system
- Score and highscore Systems
- Revive and Death Systems and Mechanics
- Added various animations for player and obstacles
- Added many UI components, score counters, death scene menus

- Voice Dictation Recognition System for adding names
- Cleaned and commented all files

Matthew Sloyan

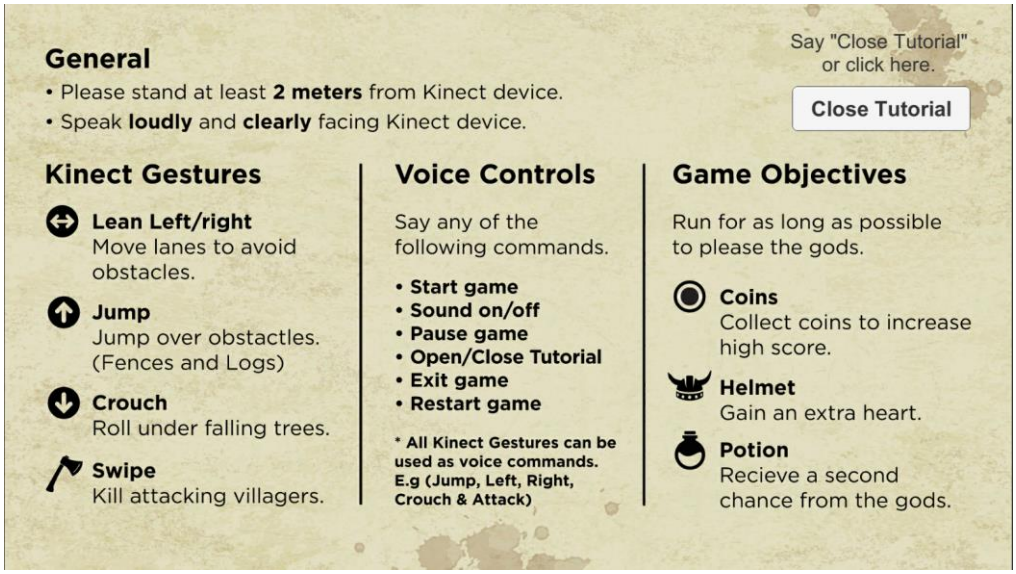
- Core game mechanics - Jump, lane movement and roll.
- Full sound system (Background music, SFX for all game mechanics, turn on/off sound etc.)
- Phrase Voice Gestures for all aspects of the game.
- Attacking NPC's plus animation work.
- Setup of random gameobject spawning (Rocks, trees, fences, logs etc).
- Improved performance on loading all objects (gameobjects and sounds).
- Tutorial and game UI design.
- Bug fixes throughout development.
- Class diagram.
- Github overview video.

Purpose of the application:

The purpose of this application is to produce a 3D endless runner in Unity which is controlled using the Microsoft Kinect V2. The game should allow the user to navigate through the game by controlling the character via a specified selection of gestures. The game should have the ability to control menu options via voice commands using Windows speech library within Unity. The UI should include a tutorial of the controls needed by the user. The reason we chose this application was to create a fun experience for the player and to improve our ability to work with hardware within game development. This is why we chose to incorporate two different methods of gestures rather than one. We also took this opportunity to work with Unity 3D as we only had experience creating games within a 2D workspace. This would also provide us with a greater challenge due to the more complex nature of 3D game development e.g. physics systems.

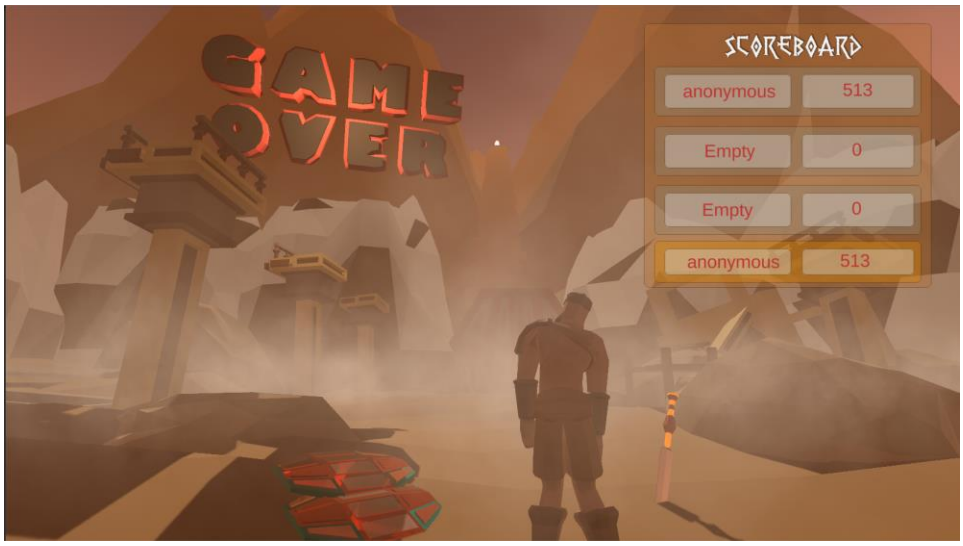
We decided that the UI should be fully voice controlled, that is removing all necessity for the user to have input controls. The user has full access to every feature in the game via voice commands. The controls for the voice commands to access the menu are detailed in the next section under the Voice Recognition heading. The user is thrown into the game where the menu is incorporated into the same scene as the game. This allows for a dynamic background making for an aesthetic view for the user.

The user is given prompts to say their name for saving their score and a prompt of how to access the tutorial with all other commands. The tutorial shows a pop up menu on screen giving the user an overview of all controls in the game.



The pause menu also can be accessed when the game is started via voice command. This pauses the character and NPC movement and pans the camera up into the sky with a dynamically moving pause menu which moves with the player.

When the player dies a dynamic death scene representative of Valhalla (Viking belief similar to heaven) is transitioned to. This also has a UI element which displays highscores, as well as the current user score.



At the beginning of the development process due to having to share technology with other groups we also fully incorporated controls using the keyboard. We decided to incorporate voice commands after we had tried and tested the kinect and ensured that they would be compatible with using the kinect. After ensuring they were compatible, we converted all the keyboards UI controls and game controls to voice commands. As a result, the game is controllable using the techniques: Kinect Gestures, Voice Control Gestures and Keyboard and Mouse Controls.

The physical Gameplay results in the player running infinitely in a direction with many obstacles getting in their way. There are 3 different chunks which spawn in the game. A village, a river and a forest. Each chunk provides different challenges for a user. A village has fences and moving NPC's (Non playable characters), A river results in the player getting into a boat and moving at an increased speed. The river has rocks which they need to avoid. The forests contain logs on the ground as well as falling trees on either side which they need to avoid. Players have many different methods to avoid these obstacles which are listed in the gestures section. There are many collectables throughout the game, coins can be

collected as a multiplier to the user score. Revives can be collected which allow the user to get a second chance when they lose all their hearts. They also can collect helmets which allow them to take an extra hit of damage without losing a heart. When the user dies their score is shown and compared to the highscores and added to the high score scoreboard if it's in the top 3 high scores.

The style and ideas of the game are based on our research. We looked at various different games and felt a 3D low poly environment would work best for our game. This was used for all elements we sources and designed. When we decided on a Viking style game we researched into how they lived and their traditions. We also researched the architecture, weaponry, clothing etc and designed our chunks, characters and the weapons using this. Another idea was the implementation of Valhalla, which was where heroes slain in battle are received by the gods. This fit well with our game so we implemented this as our game over screen which shows the player awaiting the gates to Valhalla.

Gestures identified as appropriate for this application:

Another focus of this project was to develop something that incorporated gestures but also served the purpose of being fun and easy to play. In order to achieve this we needed to incorporate simple and intuitive gestures which would allow the player to prefer using gestures rather than a standard controller/keyboard. We initially took it upon us to research different areas of gesture based technology and hardware. Some of the topics we looked at included voice recognition, full body tracking (Kinect), simple hand gestures (Leap Motion Controllers) and arm gestures (Myo). A breakdown of the hardware and their benefits can be found in the section below. Based on our research we decided to choose full body tracking and voice recognition. Firstly due to the fact our game would be an endless runner full body tracking fits perfectly with the motions that are standard to a game of this genre, such as jumping, moving and crouching. Secondly it provides a means of exercise by gamifying it. We also chose voice recognition as it goes well with Kinect gesture tracking, as the player can easily enter commands while they are playing the game without the need to pick up a controller, so it allows a full hands free experience. This also increases the safety for those around. We wanted to make the gestures as simple as possible to make it easy and fun to play. Below we will look at some of the gestures we have developed, our reasons for implementing them and our testing efforts for each.

Kinect Gestures

With the Kinect set up with Unity we began to test the inbuilt gestures, however we felt it would be best to create our own unique to our game. We decided to research the different types of gestures we could create and what the Kinect could detect, from this we decided on jump, lean right/left, crouch, and swing. We recorded all three of our bodies doing each gesture in multiple different ways to get a well rounded result. With this we had a full suite of the below gestures to incorporate into the game. We wanted to make sure they were all unique to each other, so that none would overlap. Also, we emphasised ease of play with simple gestures that anyone can complete.

1. **Jump**

To trigger this gesture the player must jump off the ground. We used a confidence level to achieve this as sometimes it could be seen as a crouch. This gesture is a simple one and even without a tutorial the player would assume this is how they would make the character jump, so we knew it was a perfect gesture.

2. **Move Left & Right**
   When the player leans left or right the correct gesture is triggered, which moves the player into the left or right lane to avoid obstacles. We tried a number of different ways of implementing this such as jumping left or right like the player does in the game. However we felt it was too much for the player to do, especially with how common it is in the game, so we went for a simpler approach. The lean gesture also worked a lot better and is more fluid with the game experience.

3. **Crouch**
   If the player physically ducks down or crouches the player will do a forward roll which doesn't break momentum. It made more sense to do a roll in the game as the player is constantly running, whereas a roll for the player wouldn't have been a suitable gesture. Based on this instead we implemented a simple crouch.

4. **Attack**
   The player can swing either their right or left arm to make the player swing their axe and kill villagers. We wanted this gesture to be seamless and not require a full body swing, the player could easily the gesture while

With these four types of gestures we have covered everything the game requires but ultimately made them all unique to each other for accuracy and ease of play.

Voice Recognition Gestures

As mentioned above to continue the hands free and ease of play experience we decided to incorporate voice recognition as our second gesture system, which merged nicely with the already developed Kinect gesture controls. We thought it would suit well to trivial tasks such as pausing the game, turning off the sound etc which could easily be spoken while jumping, attacking and leaning. From research and what we learned from lectures and labs we found that simple key words were best for detection and for people to remember. We developed a dictionary of the most common phrases for each task increasing accessibility. All of these other options can be found below along with the task. A lot of the options we added again like the Kinect gestures and very simple and could even be assumed without a tutorial which was key to our development. However we have also added a tutorial to help all players.

Also, all voice commands are implemented for the all above Kinect gestures as another means of playing the game, so it is accessible to all players. For example if a player doesn't want to or can't jump/crouch then they can use voice commands or keyboard inputs. Example inputs are as follows (jump, left, right, crouch, roll, attack, swipe).

1. **Start Game**
   Other variants - start game, play game, start, play, begin, begin game.

2. **Restart Game**
   Other variants - reset game, restart game, reset, restart.

3. **Exit Game**
   Other variants - exit game, close game, exit, close, quit game, quit.

4. **Pause Game**
   Other variants - pause game, unpause game, pause, unpause, resume, resume Game.

5. **Turn on/off sound**
   Other variants - sound, sound on, sound off, turn on sound, turn off sound, turn sound on, turn sound off, volume on, volume off.

6. **Display Tutorial**
   Other variants - open tutorial, close, tutorial, display tutorial, show tutorial, open guide, close guide, show guide, tutorial, guide, help.

7. **Name Input**
   At the start of the game the user can input their name by voice command which will be saved to the high score at the end.

8. **Game mechanics**
   Other game mechanics are implemented with voice also as mentioned above. These are as follows (jump, left, right, crouch, roll, attack, swipe).

Hardware used in creating the application:

For this application we also wanted to learn as much as possible about different gesture based technologies that are out there. This was where we started our project breaking down each technology and comparing them to each other to see what suited our game best, and what would provide the best experience for the player.

Kinect V2

The Kinect is a motion sensing input device produced by Microsoft initially in 2010. Later in 2013 the V2 model was released which is the version we researched and tested. Compared to the V1 it uses a wide angle camera with 60% wider field vision and can detect a user from 3 feet from the sensor. 6 skeletons can be tracked at once, and it can detect a player's heart rate, finger movements, facial expressions, the position and orientation of 25 individual joints along with the speed of movements. There is also a microphone available to record voice gestures and the camera records 1080p video in colour mode. Microsoft provides a Windows SDK and unity package for the Kinect V1 & V2 so it is widely accessible to developers. A number of these features were interesting to us and got us thinking of ideas initially.

Voice Recognition

There are a number of Voice Recognition Services available such as products supplied by Google and IBM, however we decided to research Microsoft's offerings due to our experience in the labs and initial project ideas. Windows provides Speech To Text (STT) by default and it can be implemented into Unity easily so this seemed like a good option for us to add useful functionality to our game. This can be added to Unity in two ways, by using a KeywordRecognizer to recognise grammar, and a DictationRecognizer to recognise words spoken by the user using STT. STT involves converting human speech to a text format so it can be read by a machine in real time. Using voice recognition seems to go hand in hand with the other technologies such as the Kinect, Leap Motion Controllers and the Myo Armband as it can be done in conjunction with these gestures.

Leap Motion Controller

The Leap Motion controller is a small USB device developed by Leap Motion which is designed to be placed on a table, desktop or virtual reality headset. It uses two IR cameras and three LEDs to observe a hemisphere around 1 meter squared. This is used to generate a 3D map of both the users hands, and can track gestures. Its capabilities include playing games, drawing in a virtual environment or scaling a virtual map along with endless other possibilities. It has some similarities to the Kinect as the Kinect can also detect finger gestures however the Leap Motion Controller provides more freedom in this regard.

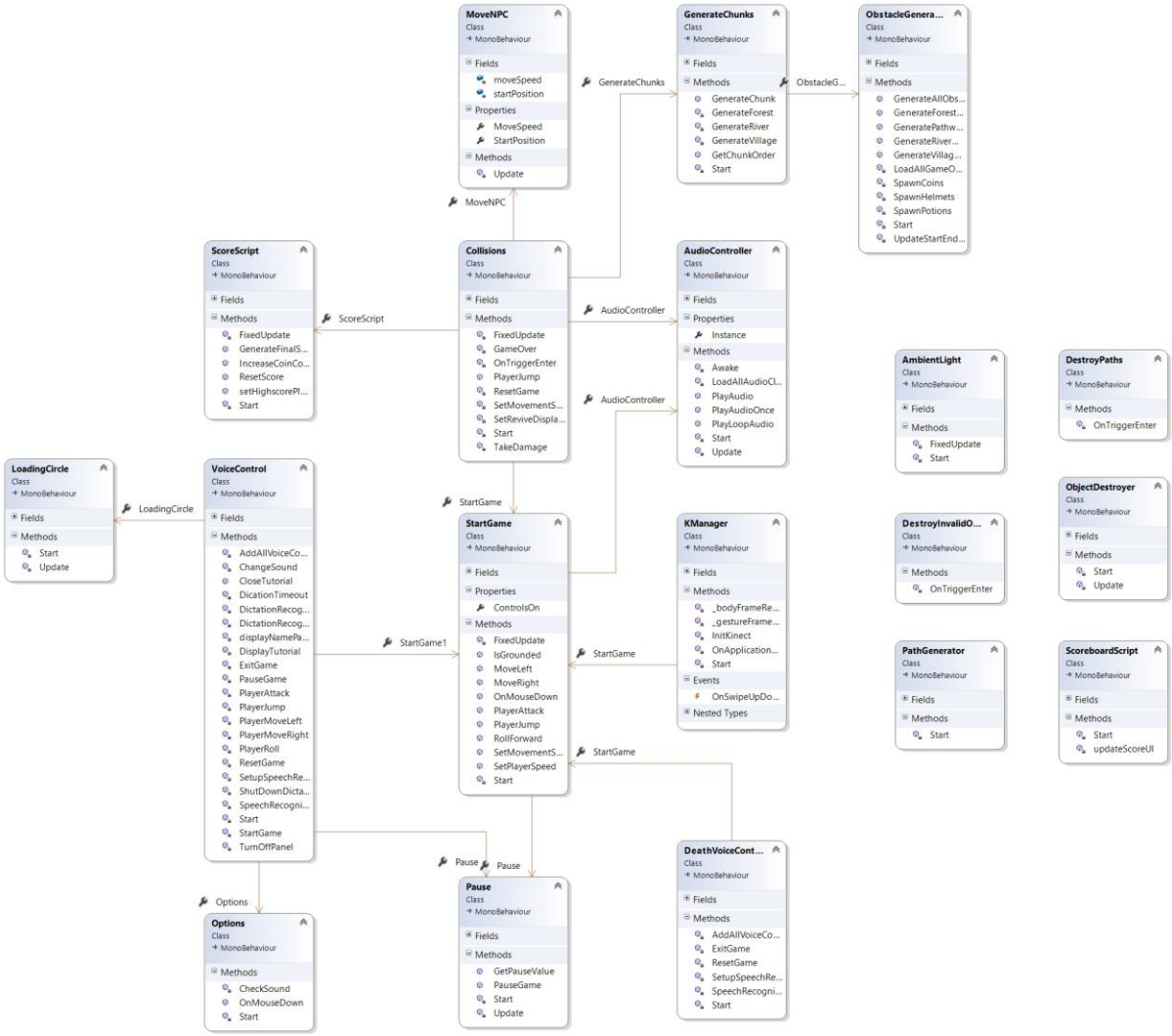
Myo Armband

The Myo is a wearable device developed by Thalmic Labs which lets you use the electrical activity in your muscles to wirelessly control your computer, phone, and other devices. Combined with a gyroscope, accelerometer and magnetometer it can recognize multiple gestures which are as follows: Wave Left, Wave Right, Double Tap, Fist and Fingers Spread. Also access to raw EMG data from the Myo is available so calculations (orientation and movement speed) can be calculated to develop your own set of gestures. It differs from the Leap Motion device as it is worn rather than a 3D array of cameras that sense motion in the environment. It is also different from the Kinect for this same purpose.

Chosen Hardware

Based on the research and tests we have chosen the Kinect V2 and voice recognition for hardware and technology used in this project. We chose the Kinect as it fitted well with our game idea and genre and would provide a fun and intuitive way to play the game. We chose voice recognition as secondary gesture technology as it complimented the Kinect well and would allow full disconnection from a device to control the game as voice commands could be entered while jumping or attacking for example.

Architecture for the solution:



Above is a layout of the flow and scripts used to create this application. A description of the main scripts, how they work and why we implemented these features can be found below.

StartGame

The StartGame script is one of the main scripts of the game and is attached to the main player. Majority of scripts flow in and out which can be seen in the class diagram. This script handles all player movement and controls. Code reusability was one of the main design choices for this script, so there is one method for jumping, moving, rolling etc which are reused and called by the Kmanger script if a Kinect gesture is detected or from the VoiceController script if a voice command is detected. Keyboard controls are also handled from this script as the third method of play, which in turn calls the same methods mentioned above. Other methods such as a check if the player is grounded are included, which was used to stop the player jumping too far. This works by sending a ray cast to the ground, which returns true or false depending on the length. E.g. if the player is on the ground or not. Lastly, the player speed and whether the controls are on or not can be set independently in

this script from other scripts using a private static variable. This is changed in the Collisions class to turn the controls on and off in the menus or at the start of the game.

KManger

This script controls the entirety of the kinect. In the start method it initializes the kinect and loads the gesture database into memory. Once the gesture database is loaded it creates Gesture objects of every gesture in the database. Once this process is done it starts to analyse each frame looking for the correct gesture. If a gesture is found and no other gesture is running, it calls the method attached to it. For example if a jump gesture is detected it calls the jump method in the startgame script. All of this allows the user to control the entire game using the Kinect V2.

VoiceControls

The voice control scripts are broken up into two parts, the Phrase Recognition System and Dictation Recognition System.

The Dictation Recognition system is used to take a user's voice input and try to determine what they said exactly. This is used for setting the user's name at the beginning of the game. The name is set and then that name is used to mark their score and highscore if necessary. The dictation recognition system is turned on upon starting of the game. When the user has said their name or the 15 second timer has run out, the dictation recogniser system is shut down. This was used as we felt the user must be allowed to enter their name without having to use a keyboard. This gives the user the freedom to choose their own name rather than choose from a predefined set of names where phrase recognition to be used..

The Phrase Recognition System is used to match user voice input to a specific phrase within a dictionary of specified phrases. This is used in the game to call methods which are linked to displaying of UI elements as well as methods used to carry out calling of methods which carry out player controls. The Phrase Recognition System starts up when the dictation recognition system is shut down, This is because unity doesn't allow both to be active at the same time. This works by adding phrases to a dictionary with an associated method to be carried out when that phrase has been detected. We added multiple options for the same voice commands to ensure the user has adaptability when trying to get their phrase recognised. E.g. the user has many options to say for a task, for example, exit, quit and close will all exit the game if recognised. This allows for users to have multiple options to carry out commands and not have to spend time learning a specific phrase for each command.

Options

This script is used to manage the game options for the user. This uses Player preferences in Unity to load in their previously saved options, e.g. music is turned off. This allows the user to not have to set their settings each time they reload the game.

Pause

The pause game script is used to create a pause game effect in the game. When the pause script is activated the camera is rotated up to the sky to view the pause menu. Rather than simply freeze time in game setting Time.timeScale to 0 which would be the easy option. We

decided to create a dynamic effect to allow clouds moving in the background and a dynamic text menu. In order to do this we individually froze all moving objects in the game by stopping just the player and NPC (non playable character) movement. When the menu is unpaused we added a timer of 3 seconds to allow the player to get familiar with their environment again before we throw them back into the gameplay.

AudioController

This script handled all the sounds for the game. Initially when started all game sounds are loaded into memory using a dictionary. This dictionary is then called when a sound is needed, so it's ready to go and doesn't need to be loaded each call which can be memory intensive. Two Audio sources were used, the first is for background music and the second is for sound effects (jump, death etc) so they can be played in conjunction with the background music and not interfere. Three methods were created also to work with both sources and the dictionary of all sounds. They have been all set up with reusability in mind as only a string is passed in a parameter, this string is used to find the audio in the dictionary which has an O(1) access time.

Using the playerPrefs set in the options script on each update it is checked if the sound is on or off, it's been turned off and all audio sources are muted. A singleton design pattern was also used to quickly get an instance of this class without searching for it, this worked also as there only needs to be one AudioController in the game so there is no overlap on sounds.

Collisions

The collisions script deals with a wide range of functionality throughout the project. It deals with all the collisions the character faces throughout the game.

The script uses the OnTriggerEnter method provided by unity to get collisions between each object, this script deals with many areas from when the player collides with the boat to jump into it and trigger animations. It deals with player damage and revive systems as well as the collectables systems.

The boat system triggers when the player collides with the boat and then locks the player into position within the boat, this is done to allow the user to progress through the river chunk.

The player damage system works by removing hearts from the players hearts when they hit an obstacle. When they collide with an obstacle with one heart remaining, then the death animation is triggered and checks to see if the player has a revive. If so the players' hearts are kept at one and they continue playing. Otherwise a transition to the death scene is made.

If the player collides with a collectable then they are either activated in the case of a revive potion or helmet (which acts as armour - extra heart for player) or they are used to increase coin count via collection of a coin. This provides more areas of risk for reward for the player as they may have to take a risk which results in their death to get a helmet or a revive which may help them progress later in the game.

The collision script also grants access to the player to be able to gain access to controls when they pass the menu screen, this is done when the player collides with a transform placed at the beginning of the first chunk. This prevents the user ruining camera

angles before the camera pans to behind the player when the game starts as well as moving in the menu system before the game has started.

The collision script also deals with many player animations by calling triggers set within the players animation controller. For example triggering death and revive animations upon players death and resurrection. These animations make the game look aesthetically pleasing as well as add an element of realism to each collision.

ScoreScript & Scoreboard script

The score script is used to increase the player score as they progress throughout the game. This does so by increasing the score when the player moves along the Z axis. When the player dies the final score is then generated. The score which they have got from progressing is multiplied by the coins they have collected throughout the level which is also calculated in this script. This is done to entice the user to take more risks in order to gain a score boost reward. When calculating the final score the user's score is compared to the list of highscores stored in the player preferences and placed into the proper spot in the highscores list if necessary along with their associated name. We decided a highscore table would be better rather than implementing a singular highscore as this may give a user a sense of achievement finishing in the top 3, rather than a singular highscore which they feel they have to beat each time. It also allows for multiple users of the game to save their scores adding a competitive edge of the game.

The scoreboard script is used to load in the highscores and display them on the screen for the user in the death screen. This loads in the playerprefernce values and sets them in their correct position in the leaderboard and with their correct name. It also displays the current user's score and name.

GenerateChunks

This script generates all the random chunks in the game. At any given time there is only 3 chunks in the scene to aid the frame-rate. If we did not destroy chunks the frame rate would bottle leaving the game in an unplayable state. We decided 3 chunks was a good balance as you could still see a good distance ahead also giving the game a good frame-rate. The script generates 3 chunks in the start method along with the obstacles. Once the player reaches the 2nd chunk the 1st is deleted and a 4th chunk is spawned and the Obstacle method is called. This process runs infinitely creating an endless runner that runs smoothly.

ObstacleGenerator & PathGenerator

This script handles all game object generation. It is used on startup for each of the three chunks (Village, River Forest), and then is called for every subsequent chunk that is generated at random from the GenerateChunks script. Initially all game objects are loaded into memory using a dictionary. This dictionary is then called when a gameobject is needed, so it's ready to go and doesn't need to be loaded each call which can be memory intensive, especially as there are hundreds of objects generated with every chunk. After this the chunk order is read in from the GenerateChunks script and the first three chunk objects are loaded. The value passed to the GenerateAllObstacles() method determines the length of the chunk (start and end). These values are used when instantiating obstacles so they start spawning at the "start" z value and end at the "end" z value. Loops are used to instantiate all objects using the "start" and "end" variable, the loop begins at the "start" and loops until it hits the

“end” at a fixed interval for each specific object. This interval was determined from testing so no object would spawn together. Also, all objects are spawned on the x coordinate in a random lane so they are different each time.

Objects that are spawned by the script include, NPC’s, fences, trees (left/right), logs, rocks, coins, helmets, potions and paths. To make sure all obstacles and paths are in their correct places DestroyInvalidObjects and DestroyPaths scripts are used which deletes all incorrectly placed objects, as from testing objects could bug out and spawn incorrectly.

AmbientLight

This script controls the Day-Night cycle. Unity by default has an ambient light variable set in every scene and it is impossible to change using an animator. Because of this it has to be changed via a script. The reason we needed to change this variable is to get complete darkness to emulate night time. There is also a directional light to act like a sun. This script changes both the ambient light and sun light. Once complete darkness was achieved, we needed to change the offset of or skydome/skybox. While decrementing the light intensity we increment the skybox’s material offset so that it changes to black imitating the night sky. This process is then reversed to change it back to day time. All of this creates a day night cycle however, the player can not see anything at night. To fix this a “firefly” appears when it is night time, emitting a yellow glow aiding the player to see.

DestroyPaths, DestroyInvalidObjects & ObjectDestroyer

These scripts are used to destroy objects which are being unused in the game. There are 3 destruction scripts in the game.

The first being the ObjectDestroyer script, this is used to destroy objects when the user is gone past them. This does this through the use of a transform which follows the player, when the transform's position is greater than the object then it is destroyed using unity's Destroy method.

The DestroyPaths script is used to destroy all the paths that spawn within a river chunk. It does this via a collider that is placed over the river, when a path collides with the transform, it is then destroyed.

The destroy invalid objects is used to prevent any obstacles from appearing in a chunk they should not. For example, if a log from the forest were to spawn into the wrong chunk, it is then destroyed. This is done if the object spawned collides with the chunk which it is invalid to.

All three of these scripts, although carried out in different ways, are all done in order to improve the overall performance and feel of the game. Leaving unused objects would significantly reduce the performance levels of the game due to the mass amount of objects involved.

Libraries used in development

- Windows Speech Library
- Kinect Unity Libraries
- TextMeshPro Unity Libraries

Conclusions & Recommendations:

To gather the conclusions we gathered together in a group call as we were unable to meet in person to discuss the project's final outcomes. In this meeting we discussed the various conclusions and recommendations which we came up with after reflecting on all the work done on the project.

This was a great project as it tested our abilities in a new area which we had no familiarities with. As a collective group we had zero experience working with the Microsoft Kinect and any of its associated scripts. We also had no experience in 3D game development. Two members of our group had experience with voice recognition in unity but not using the windows speech library provided by unity.

This new combination of new technologies provided us with a great challenge as working with new hardware will always provide many issues which a good knowledge of the inner workings of the hardware are required to develop a solution. This was definitely evident when using the Kinect. Many challenging aspects were encountered and limitations to the technology needed to be adhered to when coming up with solutions for mechanics within the game. Areas such as training gestures required even more external tools and methods to incorporate our custom gestures in the game. This provided a challenge as many users carry out the same actions in different ways e.g. some people may jump with their legs bent while others may not, so it was crucial during training to carry out the actions in as many ways as possible.

The physical scripts associated with the Kinect for unity, took a while to get our heads around. Also tutorials and documentation for the Kinect is very limited. This was great in a sense that it forced us to get stuck into the code and try to decipher for ourselves how it works and how we could adapt it to carry out the tasks we needed.

The voice controls on the other hand was more simple to get a good grasp on. Unity provides a lot of good documentation for this and it works quite well. The two types of speech recognizers we used cannot be active at the same time. This was a software limitation on what we could do. Unity only allows users to have one active at a time. So in order to use another, the whole system that was active needs to be shut down. We worked around this quite well by ensuring the user gives their name first then other voice commands are enabled. One issue with voice commands in unity is that there is a slight processing delay so commands are not instant like they are with the Kinect. The phrase recognition system must take the user voice input and match it to a specific keyword. Although still quick in a fast paced game like this, small issues such as this could be crucial to the user living or dying.

The game development proved rather tricky as 3D game development adds a load more areas for things to go wrong in comparison with 2D games which we all had previous experience with. For example, 3D physics provides a much greater challenge to get correct due to the many different aspects added in a 3D world. Collisions provided issues as well as areas such as animation creating proved to be a time consuming process to get perfect. Nevertheless, through a lot of testing we found and fixed the majority of bugs that cropped up during development and provided a game of a pretty high standard.

If we were to start this project again, firstly choosing a hardware which we had more frequent access to would be a great idea as it may have allowed us to try incorporate more advanced methods of using the Kinect, perhaps areas such as multiple players could have been looked at. Another area to look at if we had more time to develop would be to add more gestures and obstacles to the game. Also perhaps with more time, we could have created winding paths rather than straight ones. It may have been possible to have added more hardware e.g. Playstation Move controllers to act as the players hands while the kinect could control the player movement. This would provide a variation on virtual reality without the need for the expensive headset.

We certainly would recommend this project to anyone with an interest in gesture and game development as we thoroughly enjoyed our experience developing this game as well as vastly improving our research ability into developing with many different areas of hardware. Not only did this project provide us with the knowledge of working with voice controls and Kinect gestures, we also learned about many other hardwares and how they contrast in their implementation. This project provides a great understanding into working with external hardware and the cohesion process which needs to be carried out in order to combine their functionalities.

References

Kinect: https://developer.microsoft.com/en-us/windows/kinect/
Voice Recognition Unity: https://docs.microsoft.com/en-us/windows/mixed-reality/voice-input-in-unity
Leap Motion Controller: http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/
Myo Armband: https://support.getmyo.com/hc/en-us/categories/200376195-Myo-101
Viking Research:
https://www.history.com/topics/exploration/vikings-history
https://en.wikipedia.org/wiki/Medieval_Scandinavian_architecture
https://www.historyonthenet.com/viking-weapons-and-armor