Gesture Control FPS Horror/Survivor Game
Third Year Project (COMP30040)

Student: Georgios Hadjitofallis
Degree Program: BSc Computer Science
Supervisor: Dr. Steve Pettifer

The University of Manchester
The School of Computer Science

May 2015
I. Abstract

This report outlines the research into producing a first person shooter, horror/survivor game, which aims to provide a natural way of interaction with the end-user. The game is created using Unity 3D and is running on Unity game engine. It uses the Microsoft Kinect camera, in order to recognise and track the human skeleton and mirror the movements into the 3D virtual world. The Oculus Rift virtual reality headset is used to display the game and also help track the rotation of the head. The virtual character is controlled by the user, using certain gestures, which have to be recognised in real time.

Initially, the report provides an introduction to some concepts required to understand the expected behaviour of the game. Then, an extensive research for the required sensors and tools follows. The research focuses on how Unity and the code provided by third parties work. After that, information about the design and implementation of the project are provided; followed by a demonstration of the results. The last part of the report forms the conclusion, consisting of review about the outcome and feedback from users.
II. ACKNOWLEDGEMENTS

Firstly, I would like to thank my friend Nicolas Ioannides for his excellent job as primary tester. Furthermore, I would like to thank my supervisor Steve Pettifer as well as every person who participated in testing phase and provided me with honest and valuable feedback.
# Table of Contents

I. Abstract .................................................................................................................. 2
II. ACKNOWLEDGEMENTS ..................................................................................... 3
III. Introduction .......................................................................................................... 6
   A. Project Overview and Aim .............................................................................. 6
   B. Scope of the Research Investigation .............................................................. 6
   C. Tools ................................................................................................................ 6
      1. Unity Game Engine ..................................................................................... 6
      2. Microsoft Kinect Camera ........................................................................... 7
      3. Oculus Rift VR .......................................................................................... 7
      4. Blender ....................................................................................................... 7
IV. Background and Literature .................................................................................. 8
   A. Computer Science in Games ........................................................................... 8
   B. First-Person Shooter Games .......................................................................... 8
   C. Horror/Survivor Games .................................................................................. 8
   D. Alternative Ways of Interaction .................................................................... 9
   E. Natural User Interface .................................................................................... 9
   F. Kinetic User Interface ..................................................................................... 9
V. Research ................................................................................................................. 10
   A. Unity 3D ....................................................................................................... 10
      1. Unity Interface: ......................................................................................... 10
      2. Scripts in Unity: ......................................................................................... 11
   B. Using Microsoft Kinect Camera .................................................................. 12
      1. Kinect Prefab: .......................................................................................... 12
      2. Kinect Scripts: .......................................................................................... 12
   C. Using Oculus Rift VR .................................................................................... 13
      1. Oculus Prefab: ......................................................................................... 13
      2. Oculus Scripts: ......................................................................................... 13
   D. Kinect for MS-SDK ....................................................................................... 14
VI. Design .................................................................................................................... 15
   A. Project Architecture ....................................................................................... 15
   B. Game Concept ................................................................................................. 16
   C. Gesture Recognition ....................................................................................... 16
   D. Available Gestures ........................................................................................ 16
   E. Level Structure ............................................................................................... 19
      Level 1: Introduction ...................................................................................... 19
      Level 2: The dungeon .................................................................................... 19
VII. Implementation ..................................................................................................... 20
   A. Development Methodology ............................................................................ 20
   B. The Virtual Avatar ........................................................................................ 21
   C. Detecting Walk and Walk Back Gestures ....................................................... 22
   D. Gesture Controlled Avatar .......................................................................... 24
      1. Implementing walking and walking back: ............................................... 24
      2. Implementing Left and Right rotation: .................................................... 25
      3. Implementing Shooting: .......................................................................... 26
   E. Collision Detection ......................................................................................... 27
   F. Creating Enemies ............................................................................................ 28
   G. Implementing the Game Logic ...................................................................... 29
      1. Game Logic Embedded In Objects Behaviour: ....................................... 29
      2. Separating Game Logic Using Observers: .............................................. 30
VIII. Result ................................................................................................................... 31
IX. Reflection and Conclusion .................................................................................. 41
A. Reflection on Result ........................................................................................................... 41
B. Reflection on Feedback .................................................................................................... 41
C. Future plans...................................................................................................................... 41
X. References....................................................................................................................... 42

Table of Figures
Figure 1.................................................................................................................................... 10
Figure 2.................................................................................................................................... 12
Figure 3.................................................................................................................................... 13
Figure 4.................................................................................................................................... 15
Figure 5.................................................................................................................................... 17
Figure 6.................................................................................................................................... 17
Figure 7.................................................................................................................................... 17
Figure 8.................................................................................................................................... 18
Figure 9.................................................................................................................................... 18
Figure 10 ................................................................................................................................. 21
Figure 11 ................................................................................................................................. 22
Figure 12 ................................................................................................................................. 22
Figure 13 ................................................................................................................................. 31
Figure 14 ................................................................................................................................. 32
Figure 15 ................................................................................................................................. 32
Figure 16 ................................................................................................................................. 33
Figure 17 ................................................................................................................................. 33
Figure 18 ................................................................................................................................. 34
Figure 19 ................................................................................................................................. 34
Figure 20 ................................................................................................................................. 34
Figure 21 ................................................................................................................................. 35
Figure 22 ................................................................................................................................. 35
Figure 23 ................................................................................................................................. 35
Figure 24 ................................................................................................................................. 36
Figure 25 ................................................................................................................................. 36
Figure 26 ................................................................................................................................. 37
Figure 27 ................................................................................................................................. 37
Figure 28 ................................................................................................................................. 38
Figure 29 ................................................................................................................................. 38
Figure 30 ................................................................................................................................. 39
Figure 31 ................................................................................................................................. 39
Figure 32 ................................................................................................................................... 40
III. Introduction

A. Project Overview and Aim

The expected outcome of the project is a game that is able to provide an alternative way of interaction with the end-users. The game aims to leave the classic ways of interaction (e.g. keyboards, mouse, Xbox controller) and move to a natural way of interaction. In other words the end-user is no longer required to press buttons in order to play the game. Instead, he/she is able to use the motion capture sensors to mirror his/her movement from the real world to the virtual world, manipulating the virtual avatar using gestures, and interacting with the rest of the 3 dimensional virtual environment by touching other objects (when the hands of the virtual avatar collide with another virtual object).

B. Scope of the Research Investigation

The research to be conducted covers the creation of a game controlled with natural ways of interaction; focusing in the investigation and efficient implementation of the required game logic to use those natural ways of interaction in horror/survivor game. It also comments on the effect of such interaction in the user experience.

C. Tools

1. Unity Game Engine

Unity is cross-platform software development kit (SDK) developed by Unity Technologies. It is used to develop video games and it consists of a game engine and an integrated development environment (IDE).

Unity provides an easy way to develop games by creating "scenes". Scenes are where visual assets are placed in the Unity environment. A game can consist of one or more scenes linked to each other, which will be rendered and updated in real time. Assets are game objects, objects on which the developer can attach scripts to define their behaviour. The scripts define how an object will act and react with the rest of the virtual environment.

Even though there was a variety of other SDKs which can be used for the creation of a game, (e.g the Unreal Engine developed by Epic Games) Unity was chosen because it provides a powerful game engine, portability and an easy to learn and use interface.
2. Microsoft Kinect Camera

Kinect is a motion sensing input device developed by Microsoft. Kinect sensor features a RGB camera (web camera), a depth sensor (consisting of an inferred laser projector combined with a monochrome CMOS sensor) and an array of microphones (not used for this the project). Kinect is using its own build in software provides full-body 3 dimensional motion capture.

3. Oculus Rift VR

Rift is a virtual reality head set developed by Oculus VR. The goal of the company was to create a virtual reality, head-mounted, display which will be more effective and inexpensive from the ones currently in the market. The outcome was the Rift SDK, which is used for this project in order to display the game and keep track of the rotation of the head in all 3 dimensions. The rotation of the user head is fed back to the game, in order to rotate the virtual camera, giving the ability to the user to "look around" in the virtual scene.

4. Blender

Blender is a cross-platform, free and open source animation suite developed by the Dutch animation studio Neo Geo and Not a Number Technologies (NaN). Even though Blender supports the entirety of the 3D pipeline (modelling, rigging, animation, simulation, rendering), it was only used for the creation of some 3D models used in the game.
IV. Background and Literature

A. Computer Science in Games

It is a fact that game industry have become one of the most remarkable and profitable industries nowadays. Modern video games are software running on electronic devices called platforms and have become an art form and industry. All of the software utilise a user interface in order to generate visual and audio feedback to the user, which manipulates the game using an input device. This input device is referred as game controller and varies across different platforms.

B. First-Person Shooter Games

First-person shooter (FPS) is a game genre consisting of 3 dimensional games, projected through a first-person perspective. In other words the user controls a virtual avatar, placed in a 3D virtual environment, and experiences the game through the eyes of that avatar. The first ever FPS game can been traced back to 1973 with the development of Maze War. A classic example of this genre is the video game Doom, one of the most influential games in this genre, which was released in 1993 and its name was used for years to describe the particular genre.

C. Horror/Survivor Games

Survival horror is a subgenre of video games that focuses on strong horror themes. Games in this genre use the 3D environment in the form of a maze, where the user has to find his way out by fighting monsters, solving puzzles and experiencing horror animations. The prime goal of this kind of games is to surprise and scare the user.
D. Alternative Ways of Interaction

Technology was rapidly evolved on this part over the past few years. Since the appearance of the first computer, the power of the machines as well as our broad understanding of them suddenly grew, promoting the invention of innovative ideas. One of those, was the idea to develop a different way of interaction between the human and the machine. For years the only way for a human to pass information to a computer was by pressing buttons. Buttons are nothing more than pressure sensors connected to a machine, which based on what button was pressed had to perform the appropriate actions. The innovation of this technology was to use new and different kinds of sensors in order to modify the interaction between user and machine. The most famous example of such a sensor is the mouse, which made available the creation and promotion of personal computers to the public.

E. Natural User Interface

With main goal the improvement of interaction to allow effective use and manipulation of computers, new sensors eventually alter the user interface, forcing it to evolve. The new interfaces are refereed to natural, because the human-machine interaction comes naturally. Interfaces sense the physical world and remain invisible to users during the whole procedure of interaction.

One of the major game companies which invested in such interaction was Nintendo, with the creation of Wii console. Wii is able to receive physical feedback, since it can detect the position and orientation of the Wii controller. Nintendo also promote a variety of games which allow the user to interact with the game, based on this feedback.

F. Kinetic User Interface

A category of natural user interfaces is called Kinetic and, as the name suggests, uses motion capture sensors to allow human-computer interaction. The system in this case, is required to capture the position of a human part or an object and based on the motion (how and how much did the position and/or orientation change), proceed to perform the appropriate actions.
V. Research

A. Unity 3D

1. Unity Interface:

As mentioned earlier, in Unity, a game consists of a set of scenes. Each scene contains objects with scripts to define their behaviour. In other words, Unity allows the creation of a virtual environment by placing various game objects in a scene and specify the way they act and react with the rest of the environment as the game progresses. The Figure below shows Unity Editor Window for the second level, along with the seven tabs used for the development of the project.

![Figure 1: The Unity 3D interface](image)

**Scene View:** A tab that allows the user to conveniently manipulate game objects inside the scene. It is referred to as interactive sandbox and is considered one of the most important functionalities as it allows an easy way to place, select, move or remove game objects.

**Game View:** This tab provides a representation of the final game. It requires at least one virtual camera, a game object with some specific scripts which will force it to act as a virtual camera. Anything contained inside the
field of view of that virtual camera will be rendered and displayed in this tab.

**Hierarchy View:** The hierarchy lists all the game objects inside the scene. It is automatically updated every time a game object is added or removed from the scene. The game objects are displayed using tree structure in order to provide a visualisation of inheritance.

**Inspector View:** When a game object is selected all scripts attached to it will be listed inside this tab. These scripts specify the behaviour of the selected game object. Public variables can be seen, initialised and modified directly from this tab.

**Project View:** Provide a hierarchical representation of the folders and assets

**Console View:** A window used for debugging. Here warning, error and exception messages generated by Unity are displayed. It can also be used by programmers to print their own messages even while the game is running.

**Animator and Animator Controller:** Allows the preview and modification of animation clips. This clips can be used by animated game objects. Animator Controller allows to bring all the animation clips together, and either blend multiple clips to one or programmatically switch between different animations.

### 2. Scripts in Unity:

Unity allows the creation of custom scripts, which will be responsible to respond to input, arrange events and generally specify the behaviour of the game object attached to the script. These game objects can be stored as prefab in Unity, in order to reuse them and reference to them through scripts, in real time. Unity scripts use event functions. Event functions will be called by Unity, so they will pass control immediately to the script. When the function is executed, control is passed back to Unity. The main event functions used are:

- **Awake:** Called only once, when the scene loads.
- **Start:** Called once, before the first update on an object.
- **Update:** It is called in every frame, becoming the most used function. It is used to implement any kind of behaviour.
**OnCollisionEnter**: It is called when one collider / rigidbody (a script component that creates an invisible barrier of a predetermined shape and certain size around a 3D point) collides with another.

**B. Using Microsoft Kinect Camera**

Microsoft provides a prefab, a template of a game object with scripts attached to it, in order to provide a basic and clear example on how Unity can use the camera to take information, so the game can progresses.

1. **Kinect Prefab**

   The prefab used for this project consist of a virtual 3D humanoid avatar. The 3 dimensional body of the virtual avatar is hierarchically divided into bones (smaller game objects) which will be manipulated in real time.

![Figure 2: Virtual Avatar Provided by Microsoft](image)

2. **Kinect Scripts**

   There are two scripts attached to this object, one called Avatar Controller and another called Kinect Manager. The Kinect Manager script is responsible to use the Kinect Wrapper, a third script forming an interface to communicate with the build in software of the Kinect camera, requesting the position of any human like skeleton inside the camera field of view. Based on the response Kinect Manager uses the Avatar Controller functions, which try to position the virtual bones to match the movement of the human.
C. Using Oculus Rift VR

Oculus, as well as Microsoft, offer their own prefab with scripts to interact with the reality headset.

1. Oculus Prefab

The Oculus prefab consist of a game object, symbolizing the centre of all the objects, and two children objects which form two virtual cameras in a specified position and rotation from the centre.

2. Oculus Scripts

Oculus Rift prefab uses scripts to both receive and transmit information to the head set, as it has to track the rotation of the head and project the game to the user.

Camera Script: It makes the game object act as virtual camera used to render the game.
**OVR Camera Controller:** It is the main interface between Unity and the low level cameras.

**OVR Lens Correction Script:** It makes corrections to the frame before projected to the user, in order to smooth the outcome and making it projected in better way.

**OVR Player Controller:** This is the script that manipulates the virtual avatar (i.e. in this case the two cameras) to provide a first person control. By manipulation it is meant the rotation of the two virtual cameras to match the tracked rotation of the user.

**D. Kinect for MS-SDK**

The RF Solutions published to Unity Asset Store, one open source project that uses the scripts provided by Microsoft plus some custom ones, which introduce the idea of gesture recognition. The main modification performed in the Kinect Manager script, which is responsible, on every update for a frame, to check one list of gestures and update the information about a specific gesture in GestureData. GestureData is a structure holding information about the state and progress of each gesture. A gesture can be in one of three states: In progress (the gesture is still in progress), Complete (the gesture is detected) or Cancelled (gesture was cancelled and need to reset). For each frame the appropriate check for each gestured is performed, based of the old state and progress of the gesture and the motion of the user, to determine the new progress and state of the gesture.
VI. Design

A. Project Architecture

The overall architecture structure of the project can be captured by the diagram below. While the game is running on the Unity game engine, it will constantly request information from the sensors, about the position, pose and head rotation of the user and use this information to appropriately manipulate game objects in the scene. At the end of this process, Unity Game Engine produces a visual output to be displayed through Oculus Rift device, forming the state and progress of the game.

Figure 4: Project Architecture
B. Game Concept

The concept of the game is fairly simple taking the structure of a horror survivor game, where the virtual environment forms a maze. The user must control the virtual avatar in order to navigate his/her way out if the maze. The user has to unlock his/her path either by solving puzzles or react to unexpected attacks from enemies.

C. Gesture Recognition

The original idea for this part of the project was to record multiple identical copies of the set of the required gestures. In other words, record multiple identical transitions of certain parts of human body. This kind of information can be used in the game to be compared with the motions of the user, determining if a gesture occurred. The particular method was not used because of two reasons. Firstly, it required the use of sophisticated motion capture software to record the gestures, which could not be provided. Secondly it considered time-consuming, since it would require an enormous variety of records of all the gestures used in the game. These reasons led me to follow the example of gestures provided by RF Solutions, and use some provided and custom gestures.

D. Available Gestures

In order for the game to work it has to detect a set of predefined gestures in real time. The main gestures used for the development of the game are listed below. The first three gestures are provided by RF Solutions, while the last two are custom.
**Swipe Left Gesture:** The user swipes left his/her right hand in. The expected outcome is the rotation of the virtual avatar for 30 to the right direction along with the cameras.

**Figure 5: Swipe Left Gesture**

**Swipe Right Gesture:** The user swipes right his/her left hand. The expected outcome is the rotation of the virtual avatar for 30 to the left direction along with the cameras.

**Figure 6: Swipe Right Gesture**

**Push Gesture:** The expected outcome is the creation of a bullet game object at the position of the virtual camera and add a force to the bullet making it travel at the forward direction of the camera. This way user can aim using Oculus Rift headset.

**Figure 7: Push Gesture**
**Walk Gesture:** The user is walking on the spot while at least one of the hands is slightly in front of the rest of the body. The expected outcome was the forward movement of the virtual avatar with a predefined speed.

**Walk Back Gesture:** The user is walking on the spot while both of the hands are slightly behind of the rest of the body. The expected outcome the backwards movement of the virtual avatar with a predefined speed.
E. Level Structure

The game is divided into two levels, which correspond to two independent Unity scenes.

Level 1: Introduction

The first scene forms the introduction to the game. The main goal of the scene is to inform and train the user on how the kinetic interface works. The user has to successfully pass a tutorial as explained from another virtual avatar in the scene. As a result, the user after the tutorial, will be introduced to the concept of the game and be able use the gestures to progress the game.

Level 2: The dungeon

The second scene is the main scene of the game. The virtual avatar will be placed in a dark dungeon-like maze. In order to unlock the path, the user has to retrieve a secret code by killing an enemy and touch the appropriate key using the hand of the virtual avatar. After that the user has to kill two more enemies in order to acquire two keys and free the two prisoners.
VII. Implementation

A. Development Methodology

The development of the project followed the Agile process, a set of methods which follow the manifesto below.

Individuals and interactions over Processes and tools

Working software over Comprehensive documentation

Customer collaboration over Contract negotiation

Responding to change over Following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Agile Software development was used because the suggested practices follow principles which allow the focus on the real value of the software, early delivery of that value, continues improvement through a feedback loop and the response to unexpected changes in the overall project.

The main practice selected to be applied to the project was short, time boxed iterations. With this the time given for the project was divided into small chunks. Each one was two weeks long and was aiming to complete one or more features or tasks for the project. Time boxed means that if by the end of each iteration, the goals that have been planned to complete, had not finished, it did not cause the iteration be extended, but it cause the plan to change. At the beginning of each iteration fresh decisions had to be made, for both the current iteration and the general plan. The decisions where based on the changed and new requirements specified during that time.

The major requirement for the Agile methodology is the constant and continues feedback from stakeholders. Stakeholders are the costumers or potential users. For this particular project, because of the lack of costumers, stakeholders is a group consisting of project's supervisor and primary testers.
B. The Virtual Avatar

The virtual avatar, the game object controlled by the user, is a combination of the prefab provided by Microsoft Kinect and Oculus Rift communities. With the combination of those prefabs the game is able to detect the user and mirror his/her motions on the virtual avatar. Furthermore the user is able to manipulate the rotation of the virtual cameras using the Rift.

The virtual avatar in an empty scene with the default pose (calibration pose). With zero rotation in all axis is facing along the positive z plane.

![Figure 10: Player's Virtual Avatar](image)

The virtual cameras provided by Oculus Rift community where placed on the same height with the head of the virtual avatar and with orientation that provides a first-person shooter prospective.
C. Detecting Walk and Walk Back Gestures

The gestures created for the game follow the same logic with the ones provided by RF Solutions. The main properties of a gesture are the state and the progress, as explained earlier. Kinect Manager is responsible to request a check to determine how this properties change per frame.

If the gesture to be checked is the walk gesture, then the human body parts needed for check are the two feet, the two ankles, the two hands and the hip center.

The only difference between the walking and walking back gesture is the position of the hands. If both hands are behind the hip center the gesture detected is walk back. With at least one hand in front of the hip center the walking gesture will be detected.
The fragment of pseudo code below performs the check for the walk gesture, to determine if anyone of the two feet was raised and fall down which indicates the completion of the gesture.

```csharp
switch(state of the gesture)
{
    case 0: // gesture detection - foot raised up
        if (rightFootDetected && leftFootDetected &&
        rightHandDetected && leftHandDetected &&
        hipCenterDetected &&
        rightAnkleDetected &&
        leftAnkleDetected &&
        (rightHandPosition.z < hipCenterPosition.z ||
        leftHandPosition.z < hipCenterPosition.z) &&
        (Mathf.Abs(rightFootPosition.y - leftFootPosition.y)) > 0.2f &&
        (Mathf.Abs(rightAnklePosition.y - leftAnklePosition.y)) > 0.2f)
        {
            state of the gesture ++;
        }
    break;

    case 1: // gesture complete - foot fall down
        bool isInPose =rightFootDetected &&
        leftFootDetected &&
        rightHandDetected &&
        leftHandDetected &&
        hipCenterDetected &&
        rightAnkleDetected &&
        leftAnkleDetected &&
        (rightHandPosition.z <
        hipCenterPosition.z ||
        leftHandPosition.z <
        hipCenterPosition.z) &&
        (Mathf.Abs(rightFootPosition.y - leftFootPosition.y)) < 0.2f &&
        (Mathf.Abs(rightAnklePosition.y - leftAnklePosition.y)) < 0.2f;

        if(isInPose)
        {
            state of the gesture++;
            gesture is complete = true;
        }
        else
        {
            cancel gesture;
        }
    break;
}
```
D. Gesture Controlled Avatar

The software so far is able to detect a predefined set of gestures. The next step is to simulate the expected effect for each one of them, allowing the user to control the avatar. When Kinect Manager script detects a complete gesture, it calls a function to implement the appropriate effect, as described in the design section. Making the Kinect Manager to access those functions through an interface, allows the change of the gestures effect at run time. This is achieved by the usage of scripts which extend that interface and provide different implementation for the functions.

1. Implementing walking and walking back:

In order to make the virtual avatar move in the virtual environment, the Character Controller component was attach to the game object. The Character Controller is a script which provides the Move function which takes as argument the speed as a 3D vector. When Move function is called it tries to move the game object according to the specified speed.

The speed can be found by multiplying two vectors. The first one is the orientation of the game object, which can be found accessing the transform component of a game object. The second is a vector describing the required transition to each axis.

Since the game object with rotation (0,0,0) is facing in the direction of positive z plane, the z symbolise the forward direction.

So, in the case of walk, where the avatar is required to move forward, the following calculation is used.

```csharp
WalkEffect()
{
    Vector3 speed = transform.rotation * new Vector3(0, 0, 50);
    characterController.Move(speed);
}
```
While in the case of walk back, where the avatar is required to move backwards, the calculation used to find the speed changes to

```csharp
WalkBackEffect()
    {
        Vector3 speed = transform.rotation * new Vector3(0, 0, -20);
        characterController.Move(speed);
    }
```

2. Implementing Left and Right rotation:

When user uses the Swipe Left ore Swipe Right gesture the virtual avatar needs to be rotated to the appropriate direction. The use of the Rotate function is provided in the transform component. The function takes as input three float numbers, representing the angle of the rotation for each individual axis, and rotates the game object.

Rotation to the left direction implies rotation on the negative y axis

```csharp
SwipeRight()
    {
        playerGameObject.transform.Rotate(0,-30,0);
    }
```

While rotation to the right direction implies rotation on the positive y axis

```csharp
SwpeLeft()
    {
        playerGameObject.transform.Rotate(0,30,0);
    }
```
3. Implementing Shooting:

This effect was the most challenging since it required the creation of a bullet game object which has to travel in the virtual environment. The first step was the creation of a bullet prefab. A game object which can be referenced through scripts.

Below is the script used to implement the shooting effect.

```csharp
void PushEffect()
{
    GameObject cameraGameObject = GameObject.FindWithTag("Camera");
    GameObject thebullet = (GameObject) Instantiate(bulletPrefab, cameraGameObject.transform.position + cameraGameObject.transform.forward, cameraGameObject.transform.rotation);
    thebullet.rigidbody.AddForce( cameraGameObject.transform.forward * bulletSpeed, ForceMode.Impulse);
}
```

Whenever this function is called, a check in the scene is performed to detect the game object with the Camera tag. Then it makes use of the Instantiate function, which create a game object in the scene. It takes as arguments a prefab (reference of the game object need to be created), the position in the virtual world where it will spawn and its orientation. The Instantiate function returns a reference to the game object created in the scene. That reference is used to access the AddForce function in the rigibody component, which will add a force on the game object make it travel in the virtual world.
E. Collision Detection

Unity provides an easy way to detect collision between game objects in the scene. The only requirement is to attach a script with the OnCollisionEnter event function. With the addition of colliders on the hands of the virtual avatar and the use of the event function the user is able to interact with other game objects by touching them.

The code below is an example of such a function, used by the virtual enemies in this game.

```csharp
void OnCollisionEnter(Collision collider)
{
    if (collider.gameObject.CompareTag("Hand") ||
        collider.gameObject.CompareTag("Bullet"))
    {
        health-=50;
    }
}
```

The code above performs a check whenever the enemy’s collider intersects with another collider. If the other collider belongs to a game object with tag Hand or Bullet it will cause the enemy to lose 50 points from its current health.
F. Creating Enemies

All the virtual enemies are game objects which follow the same behaviour. The behaviour can be captured by the pseudo-code below.

```csharp
// At the beginning of the game
Start()
{
    detect the game object with the tag player;
}

// Whenever the frame needs to be updated
Update()
{
    if the health of this enemy is less than zero
    {
        destroy the object;
    }

    if the player's avatar comes close to the enemy
    {
        attack the player avatar;
    }
}

// On collision with another object (with collider)
OnCollisionEnter(collider)
{
    if collider belongs to a hand or a bullet
    {
        loose appropriate amount of health;
    }
}
```

When enemies attack the virtual avatar, they need to access the Health script attached to the avatar. The health script keeps track of the current health...
status. In the Health script consist a function called Cause Damage which
decrease the current health of the player according to the provided integer
argument. So the Attack function for the enemies looks like this.

```csharp
void Attack()
{
    GameObject playerObject = GameObject.FindWithTag("Player");
    Health healthScript =
        playerObject.GetComponentInChildren<Health> ();
    healthScript.CauseDamage(deallingDamage);
}
```

G. Implementing the Game Logic

Unity allows the creation of event based games, which in order to
progress the user has to perform an appropriate action, for example kill enemies
or solve a puzzle. Unity provides a variety of methods to create such a game
and for this project. Two of those methods were used to implement the logic
behind it.

1. Game Logic Embedded In Objects Behaviour:

One way to implement game logic is to place directly in the behaviour of
the game object in the scene. For example, in the first scene when the user
touches the orb rotating in front of him, the game requires the creation of a tutor
game object. This simple game logic included inside the behaviour script of the
orb game object, which is now responsible for the creation of the tutor game
object as can be seen by the pseudo code below.

```csharp
void OnCollisionEnter()
{
    if the colliding object is the player
    {
        create tutor game object;
        start the tutorial;
    }
}
```
2. Separating Game Logic Using Observers:

The second approach used, was the creation of observers in the scene. Observers are nothing more than invisible game objects, responsible to monitor the game determining if a certain actions was performed, in order to implement the game logic which allows the game to progress. The example code below, belongs to an observer which monitors the scene to find the number of zombie enemies, currently alive in the scene. When the number of enemies reach zero, this indicates that the user has killed all the zombie enemies, and the observer needs to perform the appropriate action to progress the games (i.e. in this case generate a key).

```cpp
void Update ()
{
    create a list of game objects with tag zombie;
    if the list is not initialised or is empty
    {
        create Key object ;
        Destroy this game object;
    }
}
```

Both approaches provide the same result, while they are equally efficient and acceptable. Using the first method game logic becomes part of the behaviour of the game objects inside the scene. On the other hand, with the second approach, game logic becomes the behaviour of additional game objects which monitor the game. Observers were manly used for the creation of the game, since they provide a separation of concerns, between the existing complicate behaviour of the game objects in the scene and the logic behind the game.
VIII. Result

The game is running on the Unity game engine. When the frame needs to be updated Unity uses some 3rd party scripts provided by Oculus and Microsoft Kinect community, in order to receive and transmit information from Oculus Rift and Kinect camera respectively. The Rift sensor provides information about user's head rotation. The Kinect sensor uses a build-in software to track any human-like skeleton, inside the range of its view filed, and provide 3D information about the position of skeleton parts (e.g. knees, ankles, solders). At that moment Unity game engine has to manipulate the pose of the virtual avatar to match the new pose of the user.

Furthermore, is able to determined, based on the previous and current pose of the user, if any gesture was used. The user is able to control the virtual avatar using the gestures.

Scene 1: Introduction

The user in the first scene is instructed to touch the orb floating above the terrain.

Figure 13: User touching the orb
When the user hand collides with the orb, a second avatar appears through flames.

Figure 14: Creation of the second avatar

Figure 15: The instructor avatar
That avatar will inmate and explain the gestures, used in the game one by one, to the user, who is required to perform them so the induction can finish. When the introduction finishes the first scene is cleared and the game proceeds to the second scene.

**Scene 2: The Dungeon**

In the second scene the user is placed inside a small dungeon.

![The dungeon](image16.png)

*Figure 16: The dungeon*

The user here is required to pass through obstacles and enemies to reach the first Boss.

![Path of Obstacles](image17.png)

*Figure 17: Path of Obstacles*
Figure 18: The first enemy

Figure 19: The first Boss

Figure 20: The attack of first Boss
When the user kills the first boss an orb appears. By touching the orb the password, required to unlock the rest of the path, will be reveal.

Figure 21: The orb

Figure 22: Touching the orb

Figure 23: The password
After that the user has to interact with the mushroom-like avatar in the scene.

![Figure 24: The mushroom avatar](image)

The mushroom game object will create key objects and the user is required to press the correct password.

![Figure 25: Pressing the password](image)
If the password pressed is correct the obstacle, which block the way, is removed revealing the rest of the terrain.

Figure 26: Clear path

The user is required to defeat the rest of the enemies to acquire two keys.

Figure 27: The zombie enemy
Figure 28: Final Boss

Figure 29: Final Boss Attacking
With that keys the user can free the prisoners. This will allow him to loot the treasure and clear the game.
Figure 32: The treasure
IX. Reflection and Conclusion

A. Reflection on Result

The overall outcome of the project covers all the initially decided expectations. It provides clear and well-structured code for the implementation of a fully gestured controlled game. Furthermore, it provides a description of the required game logic, which makes use of this alternative way of interaction in a horror survival game.

B. Reflection on Feedback

Most of the user feedbacks were positive since most users tend to like this kind of physical interaction with generally any kind of software. This highlights the huge potentials of games created to make use of kinetic user interfaces, even though they force the game to become tiring and indirectly more difficult to play. The feedback was used to determine the weak parts of the game which may cause the user to dislike it. The result indicate that the main issue with the game was the low accuracy in the recognition of shoot/push gesture.

C. Future plans

The main plan for this project is to improve the accuracy of the gesture recognition algorithm, since this was the main issue rising from user's and tester's feedback. The game will continue the feedback loop until testers and users are completely confident about the result.
X. References


http://electronics.howstuffworks.com/microsoft-kinect2.htm

http://wiki.etc.cmu.edu/unity3d/index.php/Microsoft_Kinect_-_Microsoft_SDK


[6] EXPANDED OCULUS RIFT SUPPORT IN UNITY by David Helgason  


[8] Kinect with MS-SDK Unity project  
https://www.assetstore.unity3d.com/en/#!/content/7747

[9] First Person Shooter (FPS) in techopedia  
http://www.techopedia.com/definition/241/first-person-shooter-fps


[12] IGN PRESENTS THE HISTORY OF SURVIVAL HORROR  
http://uk.ign.com/articles/2009/10/30/ign-presents-the-history-of-survival-horror

[13] Unity Documentation Using the Scene View  
http://docs.unity3d.com/Manual/UsingTheSceneView.html

[14] Unity Documentation Game View  
http://docs.unity3d.com/Manual/GameView.html
Siyuan Liang
[MEng] Computer Science
School of Electronics and Computer
Science
University of Southampton

[16] Unity Documentation Hierarchy
http://docs.unity3d.com/Manual/Hierarchy.html

[17] Unity Documentation Inspector
http://docs.unity3d.com/Manual/Inspector.html

[18] Unity Documentation Project View
http://docs.unity3d.com/Manual/ProjectView.html

[19] Unity Documentation Console
http://docs.unity3d.com/Manual/Console.html

[20] Unity Documentation Animator and Animator Controller
http://docs.unity3d.com/Manual/Animator.html

[21] Unity Documentation Animator and Animator Controller
http://docs.unity3d.com/Manual/ScriptingSection.html
