VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
INSTITUTE OF COMPUTER SCIENCE
DEPARTMENT OF COMPUTATIONAL AND DATA MODELING

Semester Project

Development of the 3D computer game "En Ami" using Unity game engine

Done by:
Justinas Lekavičius signature

Supervisor:
dr. Joana Katina

Vilnius
2020

Contents

Abstract

This semester project is development of the 3D computer game "En Ami" using Unity game engine, as well as utilizing motion capture solutions for animating game characters. Game engines are becoming easier to develop games with, while motion capture animation is becoming increasingly available and affordable for game developers, thus the goal of this project was to develop a 3D computer game incorporating face and body motion capture animation for more realistic presentation and immersive gameplay. Research was done on how game characters can be animated with motion capture and how this can be integrated into the Unity game engine. The Unity game engine was used to develop the game using free Unity Asset Store assets (character and environment models) and self-made 3D models, Microsoft Kinect gaming controller was used for facial and body motion capture, Microsoft Visual Studio 2019 for writing game scripts, iPi Mocap Studio for body motion capture animating, cleanup and export, and Faceshift for facial motion capture animating, cleanup and export. The result is a 3D interactive story computer game and the conclusion that a 3D game with realistically animated characters can be developed using inexpensive hardware and in satisfactory quality.

Santrauka

Trimačio kompiuterinio žaidimo „En Ami“ kūrimas naudojant Unity žaidimų variklį

Šio kursinio darbo tikslas – trimačio kompiuterinio žaidimo "En Ami" kūrimas naudojant Unity žaidimų variklį bei pritaikant judesių fiksavimo technologijas žaidimo veikėjų animavimui. Kurti žaidimus naudojant žaidimų variklius tampa vis lengviau, o judesių fiksavimo technologijos žaidimų kūrėjams tampa vis prieinamesnės ir įperkamesnės, tad šio projekto tikslas – sukurti trimatį kompiuterinį žaidimą įtraukiant veido ir kūno judesių fiksavimo technologijas tikroviškesniam vizualiam pateikimui ir įtraukiančiam žaismui. Buvo ištirta kaip žaidimų veikėjai gali būti animuojami naudojantis judesių fiksavimu ir kaip tai galima integruoti Unity žaidimų kūrimo platformoje. Unity žaidimų variklis naudotas žaidimo kūrimui, naudojantis nemokamais Unity Asset Store veikėjų ir aplinkos modeliais, taip pat naudojant savo sukurtus 3D modelius, Microsoft Kinect žaidimų valdiklis naudotas veido ir kūno judesių fiksavimui, Microsoft Visual Studio 2019 naudota žaidimo kodo rašymui, o Faceshift ir iPi Mocap Studio naudota atitinkamai veido judesių bei kūno judesių fiksavimo animavimui bei perkėlimui į Unity žaidimų variklį. Darbo rezultatas yra trimatė interaktyvi istorija bei daroma išvada, jog įmanoma sukurti trimatį kompiuterinį žaidimą su tikroviškai judančiais veikėjais, naudojant nebrangią įrangą ir išgaunant patenkinamos kokybės animaciją.

Introduction

Computer games are getting more and more mainstream nowadays, with millions of people playing computer games and the video game industry earning billions of dollars.

For reference, in 2019 alone over nine thousand computer games were released on Steam game digital distribution platform [1], both by major game publishers and by indie developers. Furthermore, developing computer games has become more accessible as well, with release of game development engines such as Unreal Engine and GameMaker Studio 2. One of such engines is the Unity game engine.

Unity is a game engine developed by Unity Technologies, first released in 2005. A vast selection of paid and free assets in the Unity Asset Store, the powerful Unity Editor and large community makes this game engine popular among both indie game developers and major video game companies. Many critically acclaimed games were developed using this engine, such as Inside, Cuphead, Cities: Skylines and even games such as Arizona Sunshine and Iron Man VR, both playable with a virtual reality headset. Games made in Unity can be either 2D or 3D, the former being made up of two dimensional game sprites and the latter composed of three dimensional models. 3D games, such as of adventure or role playing genres may have various environment models and human characters, each with their own animations which make the game world feel alive and dynamic. To make the game world feel more natural, in-game human characters need to have life-like movements and quality animations, so as not to appear stiff or robotic. The solution to this problem is motion capture technology, often utilised by major game companies in favor of traditional, time-consuming keyframe animation.

Motion capture allows an actor to capture their movements in front of a single or multiple motion capture sensors and then transform the data of the captured movement into usable animation, which can be applied to a character skeleton and then exported in a selected format to use in a virtual environment [5]. Motion capture can be used to capture either face or body movements, using either marker or markerless motion capture systems, the former using reflective markers placed on body which reflect infrared rays cast by motion capture camera back to the same cameras, in order to determine the location of the markers and the performing actor, whereas the latter type of system does not require the use of any markers or additional equipment and instead uses depth sensors for determining the distance between the camera and the actor. Although motion capture systems could have been considered a luxury years ago and only used by animation industry in Hollywood films and high-budget video games, nowadays motion capture is becoming increasingly available and affordable to enthusiasts and indie game developers. For example, Microsoft Kinect game controller or PlayStation Move controllers and PlayStation Eye cameras are generally inexpensive and can be used to produce quality animation using motion capture software, then exported to file formats such as .fbx which are supported by game engines such as the previously mentioned Unity engine. Having an interest in developing a computer game for quite a long time, as well as having experience in animating using motion capture systems I decided to develop my own computer game using the Unity game engine for computers running Windows 10.

The main goal of this semester project is to develop a 3D computer game using Unity game engine and utilizing motion capture technology for realistic game character animation. The developed computer game is intended for people who enjoy story based, slower paced games.

The tasks of the semester project are as follows:

- Analysis of similar systems (in this case similar video games)
- Selection and analysis of technologies used for development of the project
- Design of the game (theoretical part of the semester project)
- Development of the game

The final result of the semester project is a short 3D "interactive story" computer game called “En Ami”. The result can help examine how more realistic animation may impact the overall presentation and gameplay of the game, as well as determine whether it is possible to do so using inexpensive and easy to set up hardware, with no need for a special studio or expensive equipment.

1 Analysis of similar games

1.1 Introduction to the analysis of similar games

There are video games developed by various game studios which have similarities to the semester project, although differ in other terms such as selected technologies and game engines. A closer look at the similar systems and their advantages and disadvantages can help make better design and other planning choices for the developed semester project.

1.1.1 Team Bondi and Rockstar Games "L.A. Noire"

L.A. Noire is a third person action/detective game released in 2011, developed by Team Bondi using a modified proprietary Rockstar game engine and published by Rockstar Games. Over the course of the game the player solves cases, during which the player is required to investigate crime scenes, collect clues and talk to both civilians and suspects in order to obtain information vital to the investigation. The characters may not be very cooperative and may attempt to hide the truth, and that is when the player has to decide whether the character is telling the truth, omitting some details or providing information contradictory to the collected clues.
In L.A. Noire the character faces are animated using motion capture technology. L.A. Noire features motion capture technology called MotionScan, developed by Depth Analysis. Using this technology a seated actor's facial performance is recorded using 32 cameras and constructed into an animated three dimensional model which is transferred into the game.
The advantage of such technology is that its capable of capturing even the subtlest facial expressions, providing superior quality animation. That is essential to the gameplay as the player may otherwise have difficulty distinguishing character emotions and subtle expressions. High quality animation, as well as performances by Hollywood actors helps prevent the "uncanny valley" effect and deeply immerse the player into the gameplay, making the game feel like an interactive movie. However, such technology also has its drawbacks. The MotionScan system is extremely expensive [4]. The system also needs great lighting conditions and large amounts of storage to store recorded data. There are also physical limits, i.e. the actors must remain seated for the duration of the recorded take.

1.1.2 Frogwares "Sherlock Holmes: Crimes & Punishments"

Sherlock Holmes: Crimes & Punishments is an adventure computer game developed by Frogwares using Unreal Engine 3 and released in 2014. The player has to look for clues, talk to suspects and solve cases. The game is similar to the semester project in terms of the general idea and gameplay. Sherlock Holmes: Crimes & Punishments can be played both in first person view and talking to non playable characters is a core part of the game, also both games are story-oriented. Motion capture is also used in both games to animate in-game characters, however the advantage Frogwares as games studio has is access to a full performance capture system, allowing for better quality animation.

1.1.3 N-Fusion Interactive "Deus Ex: The Fall"

Deus Ex: The Fall is an action role playing game developed by N-Fusion Interactive for mobile devices running Android and iOS, and later Windows operating system. The game is also story-

heavy, with plot details uncovered via character conversations and in-game environment. Players can interact with non playable characters by engaging in dialogue, learning more about the characters themselves and the surrounding game world. The game was developed using Unity game engine and character animations, specifically during in-game cutscenes animated using motion capture. However, it appears that facial animations were done using keyframe animation. This is a disadvantage as in this game character facial movements appear to be artificial, compared to life-like facial animations achieved by other mentioned games, thus the solution used for facial animations already appears to be dated. This is understandable though, as the game was originally developed for mobile devices and presumably under a lower budget compared to other games developed in the Deus Ex franchise.

1.2 Conclusion of the analysis of similar games

The three analysed games differ in terms of used technologies, used game engines and even platforms being developed on, but they have one thing in common – they aim to present a rich, immersive story to the player. That is done not only by well written game plot, but also by presenting themselves almost like movies, going to such lengths as using Hollywood actors and incorporating their performances into gameplay, as is the case with L.A. Noire. Analysing similar games helped make better decisions concerning the overall design of the semester project. System functional requirements were set and used technologies were chosen for the semester project based on the analysis of the similar games. Since story-driven games are often best played when immersed into the game's story, a high quality graphical presentation is needed, that includes not only the graphics, but also the game world. However, animating in-game characters like in the analysed games is extremely expensive, the case being L.A. Noire for example. Furthermore, professional game development studios are known to apply state of the art technologies in their developed games, such as either in-house game engines or advanced motion capture technologies, which is simply not possible for the developed semester project. Thus, an entry-level motion capture system, specifically a Microsoft Kinect game controller and a free version of the Unity game engine will be used for development of the project.

2 Analysis of the theoretical part of the semester project

2.1 Development of animation for Unity game engine

As game animation is an important part of the developed semester project game, I have done additional research into the subject in order to determine the best practices to bring motion capture animation into the Unity game engine to be used for game character models.

2.1.1 Unity Animation System (Mecanim)

Since Unity version 4, Mecanim Animation System is included in the Unity Editor, currently simply called Animation System. Mecanim allows previewing of animation clips, transitioning them one between another and allows mapping for humanoid and generic character types, the former type featuring an appearance of a human (two legs, two arms, one head) and the latter used for other types of character models. The Mecanim system also creates an Avatar for animated characters, which translates the character model's bone structure to bone structure supported by Unity

[2]. The Unity game engine can read animation files in Autodesk FBX format and animation files have to be assigned to Humanoid avatar via Unity Editor in order to work properly with humanoid character models.

2.1.2 Accepted animation formats and compression

As mentioned, the Unity game engine uses Autodesk FBX format animation files, which is a popular format for other software such as Blender or Autodesk's own MotionBuilder. Motion capture programs iPi Mocap and Faceshift can also export animations in .fbx format but in addition to animation, the character body and face meshes and material files are included in the .fbx file as well. This is a problem for one reason, that is file size. If every animation .fbx file for both body and face animations of in-game characters also included their character model, the files would be as large as 20 megabytes each. Furthermore, it is difficult to compress them to smaller size because character model and material file compression ratio may be higher than animation file compression ratio, therefore resulting in overall higher compression ration and the compressed file taking up more space. In addition to that, Unity does not allow editing of .fbx animation files as they are read-only by default. Sometimes animation needs to be edited to clean up some jittering or artifacts in facial or body animation.
Therefore, I have concluded that the best practice for handling animation files is to "extract" the animation from .fbx files. The FBX animation files contain an animation clip file in .anim format, which is Unity's format for animation clips that are used for Unity Editor's Animator. The .anim file can be copied by highlighting it in the .fbx file in Unity Editor and pressing CTRL+D. A duplicate of the .anim format file is created, which can then be edited using Unity Editor. The .fbx file can then be deleted to save space.
The advantages for "extracting" the animation files are several. One of them is that the extracted file is no longer read-only and can be edited in the Unity Editor's Animation view by going to Window > Animation. Another great advantage is not only reduced file size, but the small compression radius of the file when compressing using, for example, 7zip. The compressed file is then significantly smaller.

2.1.3 Combining face and body animations

Face and body animations are exported using Faceshift and iPi Mocap Studio programs respectively, in separate .fbx files. At least two animations should be applied to a single character game object, one being the the animation for facial expressions and the other being the animation for body movements. To do that successfully, Unity Animation Controllers [6] are used.
An animation controller has been created for every character for the sake of convenience and organisation. The animation controllers contain States, which can be transitioned between, mirrored or looped. The States have a Motion attribute, which is the applied animation for the State. By creating two States and applying selected animations to each one, the States can then be transitioned between one another, thus allowing transition of different animations. Animation Controllers may also have different Layers for different States. The Layers can be synchronised or executed independently. For the semester project character Animation Controllers have two layers, named Face Animations and Body Animations respectively. Hierarchically the Body Animations layer is higher and the Face Animations layer is lower. I have discovered that the Face Animations layer should be lower and set for "Additive" Blending, while the Body Animations layer ought to be higher

and set for "Override" Blending to avoid animation key overriding or certain overlapping. I have been experimenting with the layer hierarchies and Blending options, only to discover glitches and artifacts in character animations. The technique I have currently chosen for Animation Controller Layers works fairly well, without face animations overriding body animations for example.

2.1.4 Advantages and limitations of markerless motion capture

For making face and body animations I have decided to use Microsoft Kinect, a depth sensor and game controller for XBOX 360 game console. Microsoft Kinect can also be used on a computer with a USB port and running Windows 7 or older operating system. The depth sensor can be used with markerless motion capture software such as iPi Mocap Studio to make realistic body animations for easy and relatively fast export (depending on the GPU of the used computer) to Unity game engine. Having such a depth sensor was one of the advantages that I wanted to make use of from the start, however I also ran into technical limitations during the course of the semester project, one of them being having to work in a small room. To fully utilize motion capture, a larger room is useful. In my case, working in a smaller room resulted in more restrained and simple animations. Furthermore, having only one depth sensor restricted the possibilities of body motion capture a bit. I was not able to perform such actions as having my hands behind my back or turning around, as the tracking procedure in iPi Mocap Studio tended to produce animation glitches. The reason for such glitches is because the depth sensor cannot capture what it cannot see, for example having arms too close to the body or behind the back causes tracking problems because of the depth sensor's inability to distinguish the location of the arms. Furthermore, quick movements cause problems for Kinect as well because of it recording video at 30 frames per second and not being able to read human motion correctly because of motion blur due to lower framerate [3]. Having at least two depth sensors may have helped increase animation quality due to depth information being recorded by two depth sensors, one in front of and the other behind the actor, allowing for more complex human movements.

2.2 System requirements

Functional and non-functional system requirements of the semester project are presented below.

Functional requirements for the project:

- Loading different gameplay sequences depending on player progress
- Acceptable quality character facial animations (crucial for enjoyable gameplay)
- Changing the game character behavior based on the actions of the players (i.e. choosing correct or incorrect options during dialogue changes game character's facial animations for appropriate response)

Non-functional requirements for the project:

- Simple to launch the game
- Self explanatory user interface
- Immersive and fun gameplay experience

The game is designed for mid-range computers running Windows 10 64 bit operating system. Currently the game has been tested on three computers, one of them being high-end, the other being mid-range and the last one being low-range and outdated. The recommended specifications for computers running the game — at least 8GB of RAM, NVIDIA GTX 1050 GPU or equivalent (or weaker dedicated GPU), Intel Core i5-7300HQ CPU or equivalent (or weaker) and around 1GB of free storage space.

The current recommended specifications have been set not only for smooth gameplay and acceptable framerates (at least 30 frames per second), but also for correct rendering of the game graphics. The game has been tested on a low-range and fairly old laptop with 4GB of RAM and Intel Centrino CPU with integrated graphics. Unfortunately, the game did not render the characters properly, making the game unplayable.

2.3 UML Use Cases diagram

The UML use cases diagram (Figure 1) represents what functions the player can perform in-game. The Player is the only actor. The player can start the main game or the tutorial section via main menu, pause the game, exit to main menu via pause screen and exit the game via the main menu. In-game the player can start and quit conversations with characters (when they are interactable), as well as select dialogue options during a conversation. When not in a conversation, the player can open or close the notebook once it is unlocked, and most importantly, the player can move their character using keyboard WASD keys and look around the environment using the mouse.

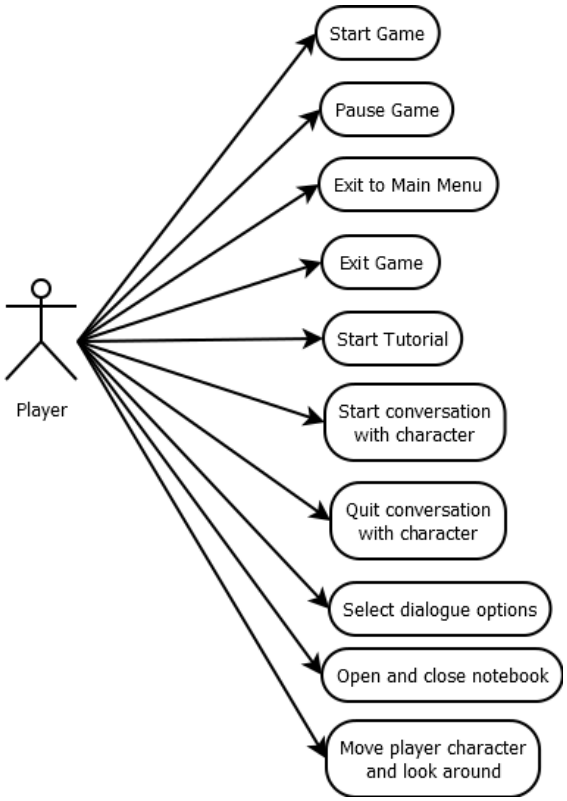

Figure 1. UML Use Case diagram of the semester project

2.4 UML State diagram

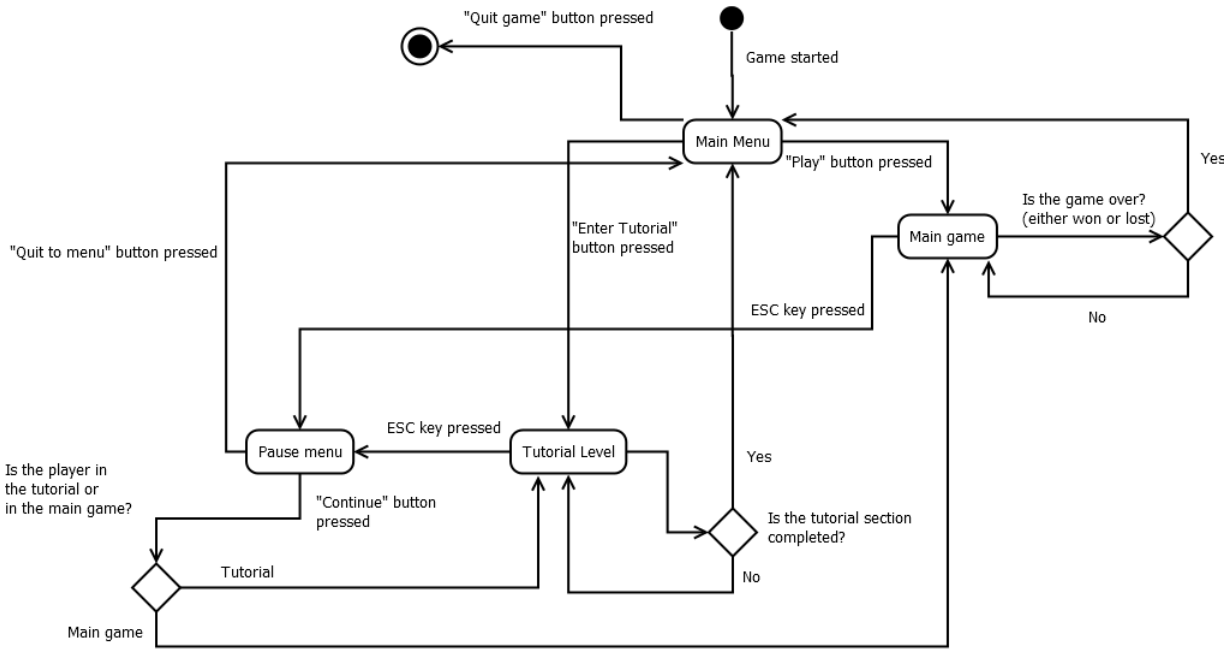

Figure 2. UML State diagram of the semester project

The UML state diagram is presented in Figure 2. Four main states of the developed system: Main Menu, Main game, Tutorial Level and Pause menu. The Main Menu is important as it allows the player to start the game or the tutorial section, transitioning into either Main game or Tutorial Level states respectively. The Main game state includes, as the names suggests, the content of the main game, including the animated characters, conversation sequences and all the other game mechanics. The Tutorial Level state is similar to the Main game state, although in smaller and more linear scale. The Pause Menu state allows the player to continue the game or go back to the Main Menu state, and can be accessed from either Main game or Tutorial Level states.

Once the game is started, the player is greeted with the main menu screen. The main screen contains three buttons, named "Play", "Enter Tutorial" and "Quit game" respectively. Should the player want to quit the game right away, they can do so by pressing the "Quit game" button. The tutorial section can be played by pressing the "Enter Tutorial" button. The player can then play the tutorial level, which presents the player some information about the game. Once the tutorial is completed, the player may quit the tutorial level. Once the tutorial level is quit, the player is returned to the main menu. The player can then press the "Play" button to play the main game. During the gameplay, either in the tutorial section or in the main game, the player can press the Escape (ESC) key on their keyboard to enable the pause menu. The pause menu contains two buttons: "Quit to menu" and "Continue game". If the "Quit to menu" button is pressed, the player is brought back to the main menu screen. If the "Continue game" button is pressed, the pause menu is disabled and the player can continue the game where they left off, either in the tutorial or the main game. Once the main game is completed, either by getting a "game over" or a "you win" ending, the player can end the main game and return to the Main Menu.

3 Analysis of the developed game

3.1 Analysis of the used hardware and software

3.1.1 Unity game engine

Unity is a game engine first released in 2005. Featuring a wide documentation [7] and a very active community of game developers, it is one of the better choices for beginner game developers. One of the strong points of the engine is its ability to build 3D, 2D, augmented reality and virtual reality games for a variety of platforms, including but not limited to Windows, Linux, PlayStation 4, Android, iOS and more. Another strong advantage over other game engines is its Unity Editor, featuring drag and drop functionality and a simple to learn over time user interface. The Unity Editor also allows to view imported models and animations, materials and edit them to a certain extent. This is highly useful, for example to clean up imported animation files or loop them seamlessly. Unity also features its own Asset Store, which allows users to download purchased and free assets. Assets range from scripts, sounds to models and animations.
The following free assets were used for the development of the game:

- Snaps Prototype | Office by Asset Store Originals
- Deucalion's Humans by Freedom's Gate
- FGC Male Adam by Freedom's Gate

3.1.2 Hardware: Microsoft Kinect

Microsoft Kinect is a motion controller developed by Microsoft for the XBOX 360 game console. The controller can used to play video games with motion controls on the XBOX 360 game console, however it can also be used with computers running Microsoft Windows for various development purposes. Although the Microsoft Kinect is designed for XBOX 360 game console, it can also be plugged into a PC with a USB port, using a proprietary adapter. The controller will be used for motion and facial capture animation. There are two reasons for using the controller. The first one is to save time while working on character animation, as acting the animations yourself and transferring them to the game engine via motion capture software is significantly easier and quicker than animating using keyframes using, for example, Blender or Unity's own built-in Animation module. The second one is to attempt to create more realistic and human-like animations, which are important to the gameplay. For example, more realistic face animations create a better game experience when the player needs to determine whether the animated character is telling the truth or they are is feeling.

3.1.3 Microsoft Visual Studio 2019

Microsoft Visual Studio 2019 is a widely used integrated development environment, used for developing programs and scripts using C#, C++, Java and other languages. Since Unity scripts are written in C# programming language, it will be used for the majority of game programming, writing code using Microsoft Visual Studio.

3.1.4 iPi Mocap Studio and iPi Recorder

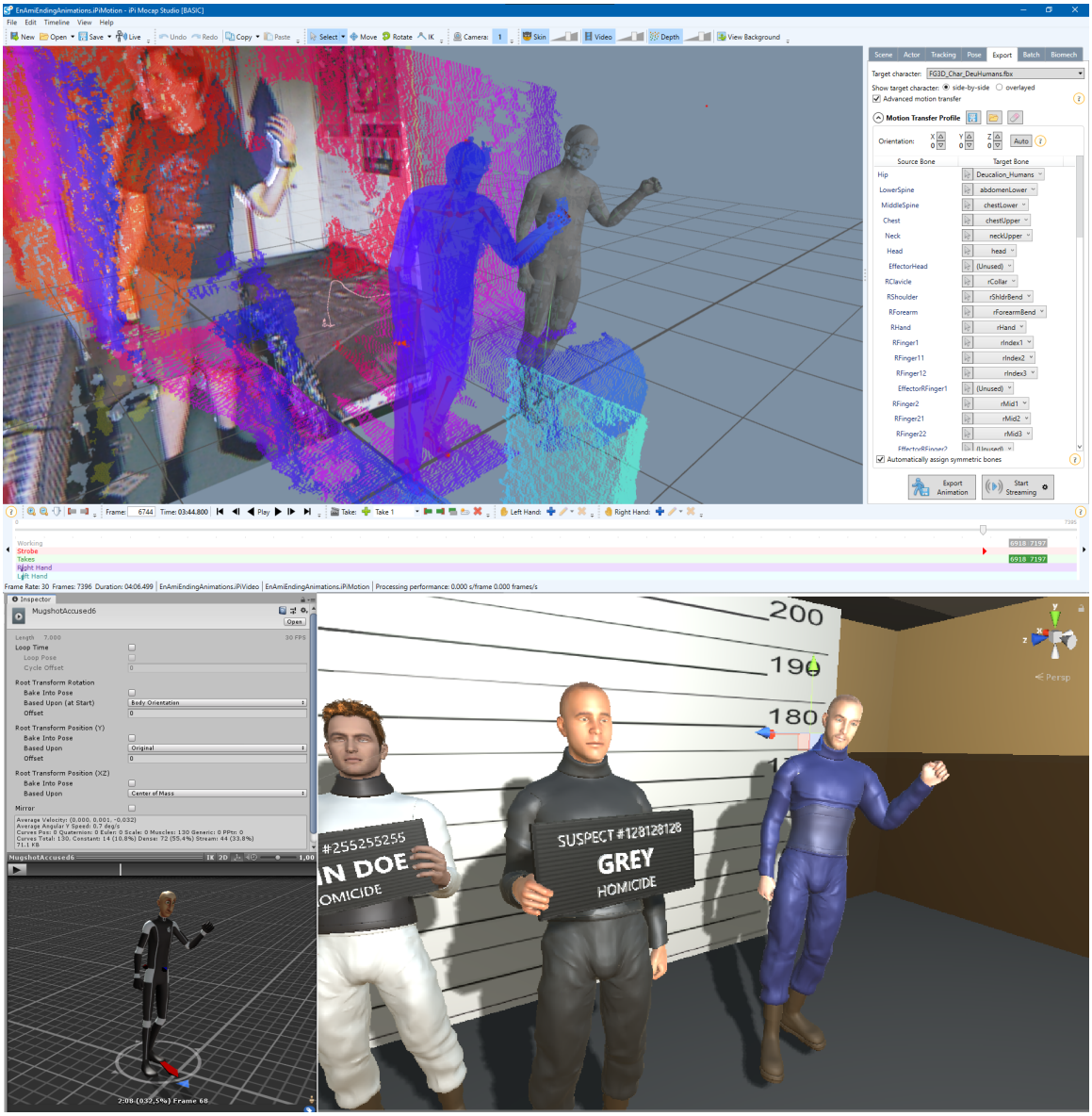

Figure 3. Top: A screenshot of iPi Mocap Studio work session. Bottom left: A screenshot of the Unity Editor's Inspector window, allowing preview of the imported animation. Bottom right: A screenshot of the game (result of the animation applied to Humanoid avatar and added to character's Animation Controller)

iPi Recorder and iPi Mocap Studio are video capture and motion capture software programs respectively, developed by iPi Soft LLC. iPi Recorder can be used to record a video of an actor's movements using a depth sensor such as Microsoft Kinect. The actor is distinguished from the background by firstly performing a background analysis, which is recording the room for up to ten seconds, then filming a motion capture session with the actor performing the movements in front of the camera. Finally the background is filtered out, with only the actor movement being processed into information for use in iPi Mocap Studio. Using iPi Mocap Studio the motion capture data can be processed into animation, with a skeleton mesh being used for tracking of capture data [3] and then used for a selected target character, in this case one of the Freedom's Gate created human character models used for the project. Once the animation is tracked, cleaned up and ready for

export, it can be exported to Autodesk FBX format for use in the Unity game engine. Animation can be exported in different formats, for example Autodesk FBX or Valve Source Engine DMX.

This software was selected for a few reasons. Firstly, the motion capture software is markerless, meaning that the actor is not required to wear any special equipment (except comfortable clothes). Therefore, it can be used at home with no need for a large studio or expensive equipment . Secondly, the software is easy to work with, supports Microsoft Kinect game controller well and is able to export higher quality animation in Autodesk FBX format, which is readable in Unity game engine. One alternative to iPi Mocap Studio could be Cinema Mocap 2, a markerless motion capture system for Unity game engine that, unlike iPi Mocap Studio, is not a separate program and can be used via Unity Editor. However, when it came to chosing between iPi Mocap Studio and Cinema Mocap 2, the former was selected due to me being more familiar and comfortable with the solution.

3.1.5 Faceshift

Faceshift is a facial capture software developed by Faceshift AG which allows a person to setup a face template and track their facial movements using Microsoft Kinect, and later export the animation in FBX format for a selected character model (or avatar). Faceshift will be used for creating in-game character facial animations.

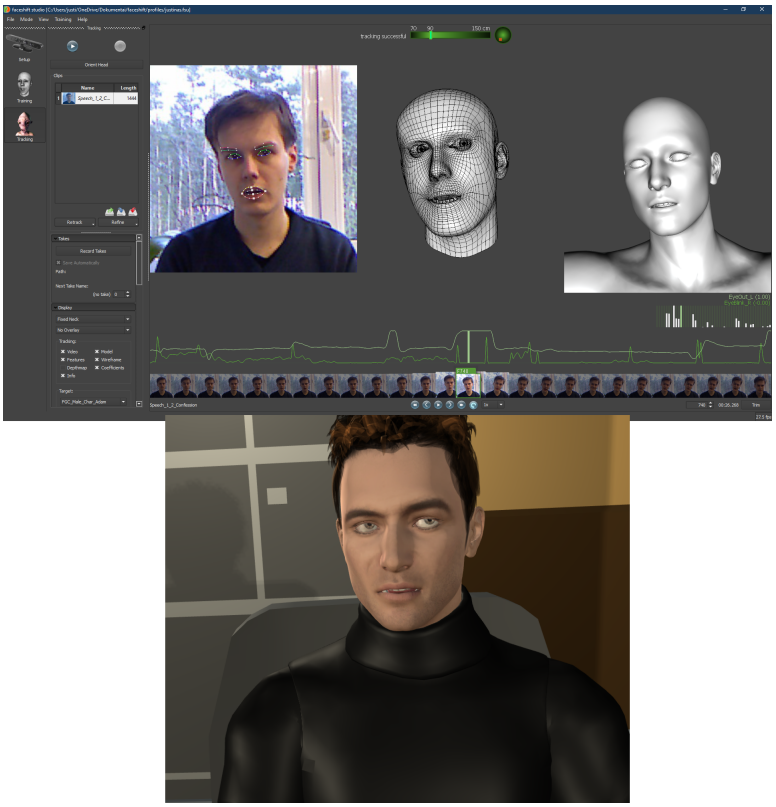

Figure 4. Top: A screenshot of Faceshift work session. Bottom: A screenshot of the game (result of the exported animation)

Before starting any animation creation, a face profile and tracking rig must be created. That is done by scanning a person's face in various expressions (smiling, neutral, sneering, frowning etc.). Once enough expressions are scanned, an actor specific tracking rig is built, which will be used

for further face tracking. Once a face profile and tracking rig are created, the user can record their performance. Once the performance is recorder, the facial movements are tracked from video and built into usable animation, which can then be refined with built-in filters. When the animation is of satisfactory quality, it can be exported for a selected character model.

This sofware was selected for the same reasons as iPi Mocap Studio, which are Faceshift being a markerless motion capture solution, easy to work with and being a personal preference with having experience working with the system in the past. Also, high quality facial animation is crucial for the developed game. As players are required to study character facial expressions in order to determine if the character is lying or telling the truth, higher quality facial animations are needed. Otherwise, the game may be difficult to play.
Alternatives to Faceshift exist as well, such as Faceware Live Client for Unity, which can be integrated into Unity Editor. One major advantage of Faceware over Faceshift is that it does not require a depth sensor such as Microsoft Kinect. However, Kinect can still be used as a webcam on Windows 10 using custom drivers and can be used with Faceware, but not in the same way as Faceshift.

3.2 Game mechanics

3.2.1 First person mode

One of the main mechanics of the developed game is moving the player character using keyboard WASD keys, pressing W to move forward, pressing A to strafe left, using S to move backwards and pressing D to strafe right. The game view can be controlled by moving the mouse. This mechanic is called the first person mode, as the player can look around in first person view and move freely like in any typical first person shooter or similar type of game.

Figure 5. A screenshot of the game in first person mode

The player controls a game object called Player, which always contains a Camera, a Cylin-

der and a Canvas. The Camera is used to render the game view. The Camera is also a parent to other game objects. For example, both in the Tutorial section and the Main Game the Camera has PlayerCinematicBars game object attached to it, with the PlayerCinematicBars object itself being a parent to two objects both called CinematicBar. The CinematicBar objects are simple models – black bars which are attached to the camera's top and bottom view, in order to simulate the effect of cinematic bars which can often be seen in Hollywood movies. In the main game the Player object also has an additional game object called ObjectiveCanvas, which contains a Text object called ObjectiveText. The ObjectiveText object displays the current objective text for the player to see in the upper left corner. The objective text can be set using SetCurrentObjective script's SetNewObjective function, which updates the objective text and also enables the display of a red exclamation mark above the current objective, for example a character that needs to be interacted to.

The Camera object has two scripts assigned to is as Components: MouseMovement and MouseVisibility. The MouseMovement script, as the name suggests, enables manipulating the Player camera. The script is written so that actually the player is rotated whenever the mouse is moved horizontally, so that the player may always move in the direction they are looking at. The MouseVisibility script toggles the visibility of the mouse cursor on-screen. By default, the mouse is disabled in first person mode, with enabled variable of bool type set to false. The ToggleMouse function in the MouseVisibility script accepts a parameter of bool type, which is then used in a logic check to hide the cursor if the passed value is false and to enable the cursor if the passed value is true.

The Player object also has a Character Controller set as a Component, which controls the Player object's height and radius for collision detection, as well as for example Slope Limit, defining what kind of slopes the player can climb when it comes to for example climbing stairs. One of the key scripts applied to the Player is the PlayerMovement script which is responsible for controlling the player character using the WASD keys, setting the movement speed and gravity (useful for climbing down stairs so that the player does not get stuck suspended mid-air).

3.2.2 The notebook

The notebook is another key mechanic of the game, providing the player useful information. The player will need to consult their in-game notebook in order to gain information about the game story and environment. The main game has game objects called Notebook and Notebook Camera, which are both moved out of bounds of the map. The notebook camera object is enabled whenever the player presses the I button while in first person mode, and the notebook is displayed. The notebook is a 3D model with different material texture set to it depending on the progress made in-game.

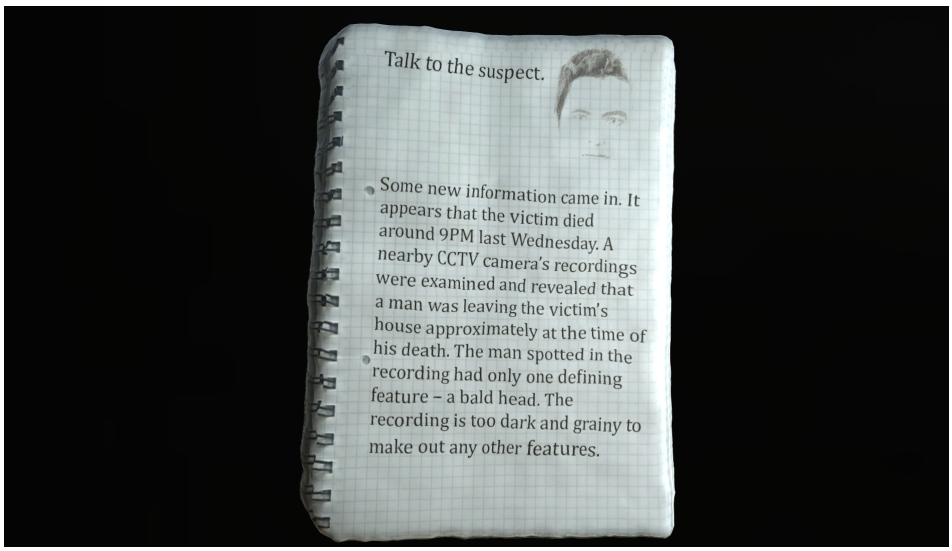

Figure 6. A screenshot of the in-game notebook interaction

The notebook has some dynamic to it, one of the aspects is its changing cover depending on the progress the player makes. The notebook cover is set using the SetNotebookCover script. The SetNotebookCover script's SetCover function accepts a Texture2D variable, which in this case is a JPG file that is used as the texture for the notebook game object's mesh. The notebook has a few different covers that display various text, in addition to the current objective which is written at the top of the notebook's page. At the beginning of the tutorial level, notebook_tutorial.jpg file is set to the notebook mesh as a texture, while in the main game a different notebook cover is set after every completed conversation (after a new objective is set).

3.2.3 Interacting with game characters

One of the most important mechanics of the game is interaction with game characters in form of conversation. Over the course of the main game some of the in-game characters can be interacted with. Once a character can be spoken to, a red exclamation mark will be displayed over them, also the objective text at the upper left corner lets the player know which character can, and needs to be spoken to. The interaction with game characters is controlled by the InteractCharacter script, which is presented below.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class InteractCharacter : MonoBehaviour
6 {
7
8     public GameObject Player;
9     public GameObject CharacterConversationCamera;
10    public bool interactable; // set to either true or false when the
          character can be interacted to.
11    public int minimumDistance = 1; //Default minimum distance before
          the player can interact with the character.
12
13
14    void Start()
```

```
15      {
16      }
17      void Update()
18      {
19          float distance = Vector3.Distance(Player.gameObject.transform.
                position, gameObject.transform.position);
20          if (distance <= minimumDistance && Input.GetKeyDown(KeyCode.E)
                && interactable == true)
21              {
22                  CharacterConversationCamera.gameObject.SetActive(!
                        CharacterConversationCamera.gameObject.activeSelf);
23                  Player.gameObject.SetActive(!Player.gameObject.
                        activeSelf);
24              }
25
26      }
27
28      public void EnableInteraction()
29      {
30          interactable = true;
31      }
32
33      public void DisableInteraction()
34      {
35          interactable = false;
36      }
37  }
```

For a game character to be interactable, the script has to be assigned to the character game object as a Component. Then, via Unity Editor Inspector screen the Player game object and Character Conversation Camera objects have to be set, as well as Interactable checkbox (interactable variable of bool type) and Minimum Distance attribute can be adjusted.

The following criteria have to be met in order for the player to be able to interact with a character:

- The player has to be close enough to the character. The minimum distance required between the player and the character before interaction can happen is adjusted by editing the minimumDistance int variable. The default is set to 1, which is one meter. The distance can be changed via Unity Editor. The distance between the Player game object and the object that the script is applied to (in this case, an non-playable character) is calculated and stored to distance variable of float type.

- The character actually has to be set as interactable. This can be done by changing the interactable bool variable to true by either the Inspector window in Unity Editor or by script function EnableInteraction.

- The E button on the keyboard has to be pressed once close enough to the character.

Once all the criteria are met, the Player game object is deactivated and the Character Conversation Camera game object is activated. The interactable characters have their own Cameras which contain multiple Canvases with different text for character speech subtitles and buttons for advancing

the conversation. On button clicks a few functions are called, including but not limited to hiding the current canvas and displaying another canvas (to progress the conversation), play a different animation for the interactable character, update the notebook once a certain point is reached in a conversation and set a new objective, as well as disable the interaction and enable the Player object once the end of the conversation is reached and the final dialogue option is clicked.
The positive of such design is that it is quite simple to control the flow of the game with button clicks. However, a huge disadvantage is that this makes the game linear. Unfortunately, I was not able to develop a dynamic conversation system due to lack of experience working with Unity game engine and developing games in general. While the current implementation works for an "interactive story" type of game, it is certainly not dynamic enough to give the game replay value.

3.2.4 The "final decision" mechanic

The final part of the main game requires the player to make a decision on which of the in-game characters is the main culprit of the crime. Of all the six game characters, one can be selected as the guilty one by pressing a corresponding button. Once a button is pressed, the character will give an appropriate response to the accusation and the player can proceed with their final decision and end the game, or go back and select a different character. Depending on the choice of the player, the game will either be won or lost, as either an innocent or guilty man will be put behind bars.
The final sequence is presented as a game object called EndingSequence, which contains other game objects such as all six characters, several environment props and cameras for each character. The game object is activated once the player completes the final conversation and is ready to make the "final decision". Six buttons are presented for selection of each of the characters as the killer. For example, clicking the button titled "Blue" will activate BlueAccuseCamera object and play a different animation for the character. The original idea was to have a dynamic ending with the guilty character changing with every playthrough, however due to lack of a clear idea how such system ought to be implemented, the thought was scrapped and instead, every time the player plays the game, the outcome will always be the same with the same character being the main culprit. This unfortunately makes the game more linear.

3.3 Game character animations

One of the most important parts of the developed game is game character animations. Over hundred diferent animation clips total are applied to the six in-game characters for diferent in-game situations. Characters are animated using Unity's Animator Controllers, which have States with applied animation clips to them. The Animator Controllers are assigned to character objects via Unity Editor.

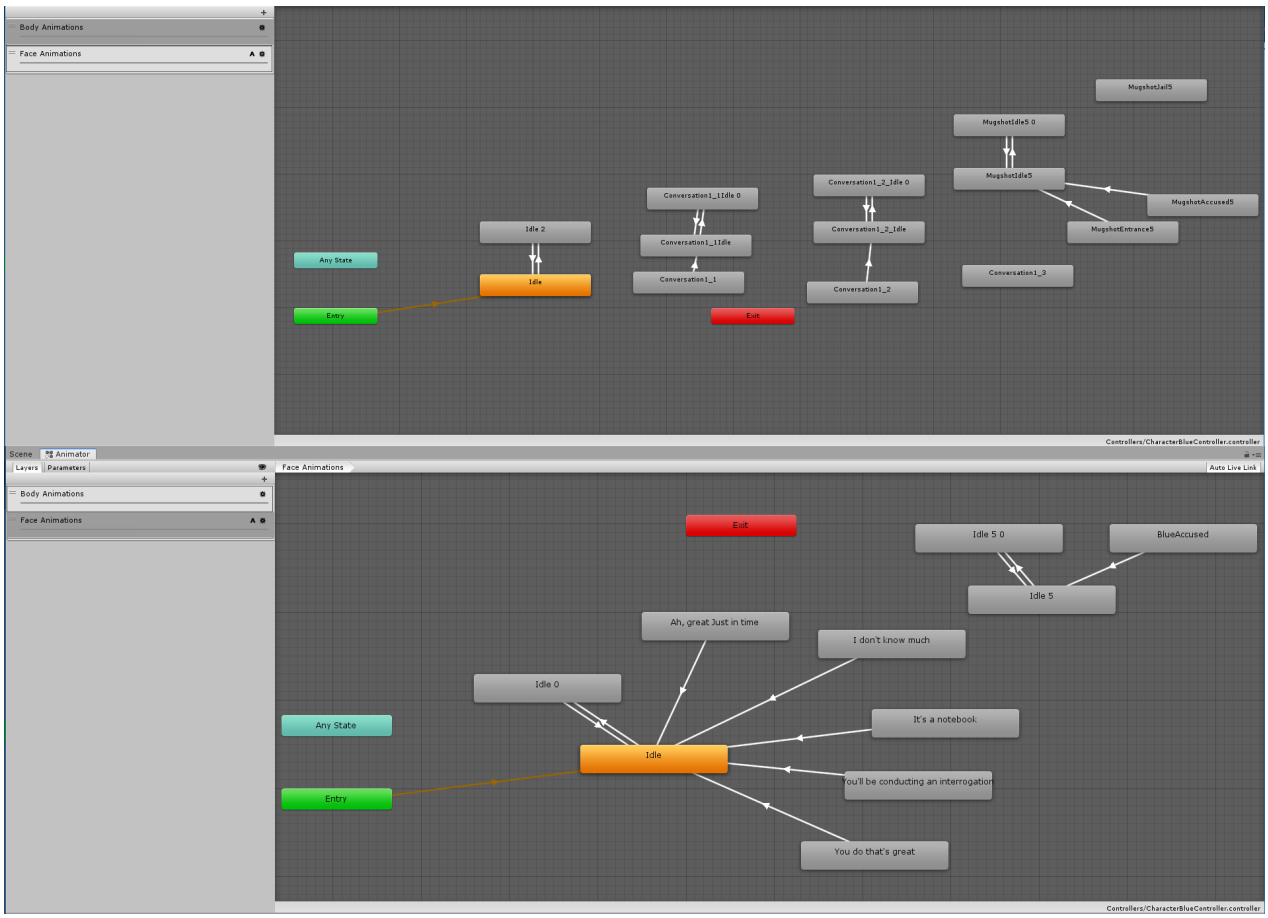

Figure 7. A screenshot of one of the Animator Controllers for the character Blue, displaying Animation States for layers Body Animations and Face Animations.

Eight Animation Controllers were created, seven for the six characters (one for characters Blue, Red, Yellow, Green, Grey and two for character Suspect), and one for animating the camera in the Tutorial level. Each Animation Controller has several States that have animation clips assigned to them. The default State that plays first is the Idle state, which contains either the character's face or body animation in idle form, making the character simply stay in place and look around. Sometimes other States are transitioned into the Idle state. This makes the character, for example, transition into their idle animation after finishing a sentence. In the case of face animations, there are three States that are mostly transitioned to from other states, those are called Idle, Calm and Nervous. The Idle State has an animation of the character simply looking around assigned to it, the Calm State, once played, has the character, similarly to the Idle State look around a bit or simply maintain a subtle calm look. The Nervous State is transitioned to mostly from States that play animation clips of characters telling lies. The Nervous State contains animation clips of character facial expressions displaying unease, looking around either nervously or suspiciously and generally being less subtle. The three mentioned states help smoothly transition character face animation, for example avoiding eye contact after telling a lie, or keeping a straight face after telling the truth. This helps the player determine whether the character is lying during the conversation or not.

Character Animator Controllers have two layers, Body Animations and Face Animations. The Body Animations layer contains Animation States for body movements, such as characters either just standing idly, walking a bit forward or performing other body expressions. The Body Animations layer is hierarchically higher and has Blending set to Override. The reason for that is because

the Body Animations layer's States have animation clips assigned to them that animate the whole body of the characters, except for their face. Meanwhile, the Face Animations layer is hierarchically lower than the Body Animations layer and has its Blending set to Additive. That is because this layer's States have animation clips assigned to them that only have facial and head/neck animations, while the rest of the body is not animated. The Blending for this layer is set to Additive as opposed to Override because Face Animations layer needs to add information from its animation clips to the information of Body Animations layer animations, thus animating both the character's body and face at the same time. If Override Blending were set to the Face Animations layer, the information of the Body Animations layer would be ignored and only the character's face would be animated, with the rest of the body being stuck in a T-pose, which is the default pose for characters with no animation.

Conclusions and Recommendations

The result of this semester project is a developed short 3D "interactive story" game with realistic character face and body animations applied to in-game characters for enhanced presentation, immersion and gameplay. However, due to technical difficulties and lack of experience in game making prior to starting the semester project I was unable to complete my vision for the game, thus it ended up shorter than I expected, and lacking in content as well as being linear. While research on both the Unity game engine and the possibilities of motion capture was done throughout the duration of the development of the semester project, sadly it did not help fully realise a perhaps a bit too ambitious idea for a beginner in game development. Furthermore, focus on quality of the presentation resulted in less quantity of the game content. The project could further be improved by introducing more dynamic game mechanics, such as different game endings and more unique in-game character conversations, providing the developed game higher replay value and enjoyment to the player. Nevertheless, it is my conclusion that it is indeed possible to develop a computer game using an entry-level motion capture system for body and face animations without need for a large studio or high-end equipment, given that one has more time and experience working with Unity game engine and development of computer games in particular. I discovered that it is not particularly difficult to import motion capture animation into the Unity game engine and apply it to game characters, however the whole process may take a long time, at least that was the case on my end. Nonetheless it is not a perfect solution, sometimes resulting in mediocre quality animation. Should a game be developed using a similar setup to the one used for the semester project, my recommendation would be to use a dual depth sensor solution for body motion capture. Two depth sensors are more likely to yield better results, as working with one Kinect controller limited the potential of high quality motion capture body animation.

References

[1] Christina Gough. Number of games released on steam 2018. https://www.statista.com/statistics/552623/number-games-released-steam/, 2019.

[2] Linus Håkansson and Filip Larsson. Developing a workflow for cross-platform 3d apps using game engines. 2013.

[3] Viktor Mattsson and Timmy Mårtensson. Viability of using markerless motion capture: In the creation of animations for computer games, 2014.

[4] Paula Pszczoła and Radosław Bednarski. Creating character animation with optical motion capture system. *Computer Game Innovations*, pages 204–218, 2016.

[5] M Rahul. Review on motion capture technology. *Global Journal of Computer Science and Technology*, 2018.

[6] Unity Technologies. Animation controllers. https://docs.unity3d.com/2018.4/Documentation/Manual/AnimatorControllers.html, 2020.

[7] Unity Technologies. Unity user manual (2018.4). https://docs.unity3d.com/2018.4/Documentation/Manual/index.html, 2020.