**Diploma Project  
Impulse   
Third person roleplaying game in Unity**

A picture containing green

Description automatically generated

# Abstract

The goal of this project was to produce a basic 3D game with movement controls and combat capabilities, including collision detection. This was accomplished using the development environment Visual Studio, in the programming language C# and the game development platform Unity. To keep the project within a reasonable scale, some demarcations were needed, such as there being no hostile foes, which is common within the genre, and the combat system remaining rudimentary. Implementing collision detection was challenging and is explained in higher detail than other parts of the process. Detection of collisions was managed using spheres, cast along a line, which check if they are colliding with the target. With the game working as intended, the project is considered successful. Furthermore, I have concluded that familiarity with Visual Studio and a basic understanding of the programming language C# is helpful when creating videogames within the Unity environment.

# Sammanfattning

Projektets mål var att, med utvecklingsmiljön Visual Studio, i programmeringsspråket C# och genom spelutvecklingsplatformen Unity, producera ett enkelt 3D-spel med rörelsekontroller och som kan hantera strid och kollisionsdetektering. För att hålla arbetet inom rimliga gränser behövdes några avgränsningar. Exempelvis finns inga fientliga motståndare, som ofta återfinns inom genren, och stridssystemet förblir grundläggande. Att implementera kollisionsdetektering var en av processens mödosammare delar och detaljeras därför mer ingående än andra aspekter. Kollisionsdetekteringen hanterades med hjälp av sfärer som projiceras ut i en linje och kontrollerar om de kolliderat med målet. Då spelet fungerar som avsett betraktas projektet som lyckat. Jag har även kommit fram till att erfarenhet i Visual Studio och grundläggande kunskaper i kodspråket C# är till stor hjälp vid speltillverkning i Unity-miljön.

# Table of contents

[Abstract 1](#_Toc102076750)

[Sammanfattning 1](#_Toc102076751)

[Table of contents 2](#_Toc102076752)

[Introduction 3](#_Toc102076753)

[Purpose 3](#_Toc102076754)

[Questions studied 3](#_Toc102076755)

[Method and materials 3](#_Toc102076756)

[Results 5](#_Toc102076757)

[Programming 5](#_Toc102076758)

[Movement system and combat 5](#_Toc102076759)

[Detecting Collisions 6](#_Toc102076760)

[Discussion and conclusions 7](#_Toc102076761)

[References and sources used 8](#_Toc102076762)

[Appendix 1 – game example images 9](#_Toc102076763)

[Appendix 2 – code examples 12](#_Toc102076764)

[Appendix 3 – Visualization of collision 14](#_Toc102076765)

[Appendix 4 – further description of collision detection 16](#_Toc102076766)

# Introduction

Roleplaying games have been a widespread form of entertainment for a long time and remains one of the most popular game genres. The format is commonly combined with another common category: adventure games.[[1]](#footnote-2) Together these two genres form the basis of popular titles such as Diablo, World of Warcraft, Cube World, Dark Souls, et cetera. The purpose of the game created in this project was the ability to serve as the foundation for a complete game of the same type.

## Purpose

Producing a working third person 3D game containing movement, camera controls and combat within Unity[[2]](#footnote-3) using the programming language C#[[3]](#footnote-4).

## Questions studied

How can the knowledge accumulated from the courses Programming 1 and Programming 2 be applied when working within the 3D Unity environment?

How can collision detection effectively be implemented in a 3D Unity game?

## Method and materials

The program and programming language used to write, test and debug for code within the project were Visual Studio 2019[[4]](#footnote-5) and C#, respectively. Unity was the only game engine used throughout production. Reasons for deciding to utilize Unity include having been exposed to C# and Visual Studio during the course Programming 2, both of which are used in the program, and its apparent ease of use.

All models, textures and animations used in production of the game were supplied by Lucas Andersson Lundvik, student at the programme of technology at Karlfeltgymnasiet, Avesta. Sound design and dialogue was produced by student Isak Siltala, of the programme of technology at Karlfeltgymnasiet.

The preliminary stages of the project included the making of a series of prototype games, the purpose of which were to become familiar with the environment and find solutions for simple issues. The first two of these games were small, 2D projects in which the player, a cube, could move across a side-scrolling[[5]](#footnote-6) world and jump past obstacles. The next project was a simple 3D game where the player, a cylindrical shape, was able to jump and perambulate on a small platform and a set of stairs. Using the players mouse inputs to swivel around them, a third person camera focusing on the player character would determine the appropriate direction to move the player using WASD[[6]](#footnote-7), or arrow key inputs. In further development of this project, proper models, animations, and scenery were created and included. Additionally, a more sophisticated movement system was introduced in which the player could hold down a key to sprint, increase their speed, and restrictions being placed on the frequency of jumps. (See images one through three in appendix 1.)

With the knowledge gained from these test games, production of the final project was initiated.

# Results

The method has resulted in a fully functional 3D third person game, created in Unity, which includes complete movement and combat integration, including collision detection. The project was, as such, successful in completing its objective within the established parameters.

### Programming

Coding within the project was conducted in Visual Studio 2019, using the programming language C#.

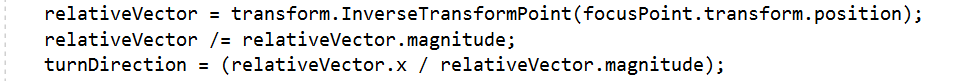
The inputs used to control the character are processed by a class named InputManager. The class receives and manages inputs from the player. It is designed to work not only using mouse and keyboard inputs, but also has the capability of accepting other inputs such as those from a controller, or mobile device. (See first image in appendix 2.)

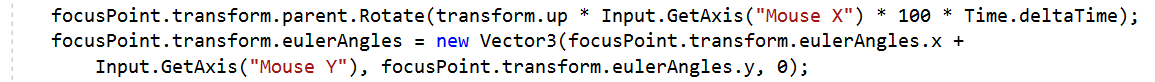
#### Movement system and combat

The system for controlling player movements, including the combat system is operated through the class ThirdPersonMovement, which receives inputs from the previously mentioned class InputManager. The class initially creates the weapon which the player wields, a simple sword, and places it in the appropriate location, either at the player characters side, or in their hand, depending on whether the weapon is sheathed or in use. When the player is holding the weapon, they are considered in the state “combat”, and while not wielding it, their character state is set to peaceful. These states are later used to determine what actions the player can perform.

A function, simply named Movement, which is called with every update, performs the calculations and logic for how the character moves in the world, including artificial gravity, the ability to jump while standing on the ground, and moving in the horizontal plane. To determine if the player is on the ground, a sphere[[7]](#footnote-8) is projected beneath the player object, checking if it intersects another object of the type “Ground”. If it does, the player is deemed “grounded”, and if it does not, the inverse is considered true.

The camera object, which rotates around the player, facing in the players direction, is controlled with the function MouseLook, which is called in the Update function. To calculate the angle and position of the camera, a few lines of code are employed within.

Further, the movement of the camera in accordance with mouse inputs is also managed within the function.

The attacking sequence is initiated from within the ThirdPersonMovement class, in the function LightAttackCombo. The function controls when and how the player is able to attack, and sends the appropriate information to another class, MeleeStats, in which the collision detection is processed.

#### Detecting collisions

The method used to detect collision of the weapon involved spherecasts[[8]](#footnote-9) interacting with pre-defined colliders[[9]](#footnote-10), and is calculated inside the MeleeStats class using the WeaponCheck function (see second image in appendix 2). A sphere is projected from either end of the weapon’s blade, moving along it to a defined maximal distance. If either of the spheres contacts an object of the type “enemy”, the target in question is notified that it has been hit by an attack and receives damage.

A further description is provided in appendix 4.

# Discussion and conclusions

The project has been a challenge and a learning experience, as I, prior to starting it, had no experience working with Unity. Most parts of the code required multiple iterations and substantial amounts of time spent scouring the Unity website for information on various Unity-specific functions. I did find that Unity was, indeed, a suitable environment to create games in, given the prior experience I had with C# and Visual Studio.

The familiarity with Visual Studio and understanding of C# gained during the course Programming 2 was massively helpful in making up for my inexperience using Unity. Knowledge of how to properly utilize classes, functions and variables was crucial in making the project achievable. Still, the experience did not carry over as much as I had hoped, with many things working differently in Unity than they did in Monogame[[10]](#footnote-11), the framework used in the course Programming 2.

During experiments trying to find a suitable method for detecting the collision of game objects, a few methods were tried and discarded before finding one that fit the project’s needs. One promising candidate revolved around encapsulating the object with a capsule collider and checking if the capsule intersects another collider. While this approach was successful in detecting collisions, it was ultimately discarded as no means were discovered by which the data of the intersecting object could be virtually manipulated. In an earlier iteration of the system currently used, a ray was projected through the object, which would react if it collided with a designated target. This method was functional, though it was deemed unideal, as the ray did not properly reflect the width of the object. In the final iteration, spheres are projected along the object, reacting to collisions with a pre-determined set of targets. Utilizing this approach, the size of the object can be properly reflected in the collision records, and the targets data can be manipulated effectively.

If the project were recreated, the game development environment Unreal Engine[[11]](#footnote-12) would be more carefully considered as an option, replacing the role Unity had within the project. The reason for this is the newly released, highly advanced and immensely powerful “Unreal Engine 5”[[12]](#footnote-13) game engine. I believe that familiarizing oneself with such a modern and robust development tool would be a substantial boon moving forward within game development.

Sources of error in the current iteration of the game include:

1. Though the game has been thoroughly tested and controlled, some bugs may still be present.
2. It is unclear how the game would behave if it were run on a less advanced computer, as it has only been tested on modern hardware.
3. As it has only been tested on Windows 10 computers, it is unclear whether it would work correctly on older or newer operating systems.
4. Operating systems other than Windows have not been examined. There is thus no guarantee that it would run appropriately on machines utilizing software of this kind.

# References and sources

Epic Games. (2022, April 29). *Unreal Engine*. Retrieved from Unreal Engine: https://www.unrealengine.com/en-US/

Epic Games. (2022, April 29). *Unreal Engine 5*. Retrieved from Unreal Engine: https://www.unrealengine.com/en-US/unreal-engine-5

Microsoft. (2022, April 26). *C# documentation*. Retrieved from Microsoft Build: https://docs.microsoft.com/en-us/dotnet/csharp/

Microsoft. (2022, April 26). *Visual Studio*. Retrieved from Visual Studio: https://visualstudio.microsoft.com/

Monogame. (2022, April 29). *Monogame*. Retrieved from Monogame: https://www.monogame.net/

Statista. (2022, April 27). *Video game preferences by genre in the U.S. in 2022*. Retrieved from Statista: https://www.statista.com/forecasts/997151/video-game-preferences-by-genre-in-the-us

Unity. (2022, April 28). *Collider*. Retrieved from Unity Documentation: https://docs.unity3d.com/ScriptReference/Physics.Collider.html

Unity. (2022, April 26). *More than an engine*. Retrieved from Unity: https://unity.com/pages/more-than-an-engine

Unity. (2022, April 26). *Physics.CheckSphere*. Retrieved from Unity Documentation: https://docs.unity3d.com/ScriptReference/Physics.CheckSphere.html

Unity. (2022, April 28). *Physics.SphereCast*. Retrieved from Unity Documentation: https://docs.unity3d.com/ScriptReference/Physics.SphereCast.html

# Appendix 1 – game example images

Chart

Description automatically generated with low confidence

Graphical user interface, text

Description automatically generated

A picture containing plant, green, leaf, indoor

Description automatically generated

# Appendix 2 – code examples

En bild som visar text

Automatiskt genererad beskrivning

En bild som visar text

Automatiskt genererad beskrivning

Appendix 3 – Visualization of collisionA picture containing plant, decorated

Description automatically generated

**A picture containing green, outdoor object

Description automatically generated**

# Appendix 4 – further description of collision detection

From the ends of the object that manages the detection, in the case of this project, a sword, a sphere is projected in a line, to a maximal distance, in this case the length of the blade (see first image in appendix 3). Upon colliding with an object of a specified type, the sphere’s distance from its origin is recorded. If this distance is not the maximum distance which the sphere could be projected, the attack is considered a hit. For this method to work properly in its role, it was necessary to cast one sphere from either side of the blade. The reason for this is that a spherecast would not consider itself to be contacting a collider if its origin was inside the same collider (see third image in appendix 3).

Because the class is a child object to the sword itself, the origin, size, and direction of the spherecasts can be manipulated to match the blade (see images in appendix 3).

When the function is called by the LightAttackCombo function, targets are continuously searched for during the swinging of the sword. If an enemy collider is hit, a function linked to the enemy object is called, which is responsible for making the enemy in question receive damage and play an animation.

1. <https://www.statista.com/forecasts/997151/video-game-preferences-by-genre-in-the-us> (accessed 27th of April 2022) [↑](#footnote-ref-2)
2. <https://unity.com/pages/more-than-an-engine> (accessed 26th of April 2022) [↑](#footnote-ref-3)
3. <https://docs.microsoft.com/en-us/dotnet/csharp/> (accessed 26th of April 2022) [↑](#footnote-ref-4)
4. <https://visualstudio.microsoft.com/> (accessed 26th of April 2022) [↑](#footnote-ref-5)
5. Side-scrolling video games are games in which the one views the action from a side-view camera angle, and the screen follows the player as they move left or right. [↑](#footnote-ref-6)
6. WASD refers to the W, A, S and D keys, which are commonly used in place of arrow keys to control player characters in videogames. [↑](#footnote-ref-7)
7. <https://docs.unity3d.com/ScriptReference/Physics.CheckSphere.html> (accessed 26th of April 2022) [↑](#footnote-ref-8)
8. <https://docs.unity3d.com/ScriptReference/Physics.SphereCast.html> (accessed 28th of April 2022) [↑](#footnote-ref-9)
9. <https://docs.unity3d.com/ScriptReference/Collider.html> (accessed 28th of April 2022) [↑](#footnote-ref-10)
10. <https://www.monogame.net/> (accessed 29th of April 2022) [↑](#footnote-ref-11)
11. <https://www.unrealengine.com/en-US/> (accessed 29th of April 2022) [↑](#footnote-ref-12)
12. <https://www.unrealengine.com/en-US/unreal-engine-5> (accessed 29th of April 2022) [↑](#footnote-ref-13)