Practical Gaming 2024

# Name of Student

Kass Smith

# T Number

T00240394

# Name of Project

Knights Night Parkour

# Gameplay

The game is played with basic WASD movement, SPACE to jump, and the mouse is used to move the camera.

The objective of the game is to get to the end of the parkour course without falling. There will also be other obstacles in the way such as moving walls and platforms, giant axes, and canons. There are checkpoints on the way as to not lose your progress, just walk into them and you will spawn back at your most recent checkpoint

# Coding

* **Frame Rate Independence**
  + Frame Rate Independence ensures code behaves consistently regardless of the frame rate. Higher frame rates shouldn’t make objects move faster or slower.
  + Frame Rate Independence is crucial for a video game. Frame rate can vary depending on hardware and complexity, so without frame rate independence, movement and animations would become jittery at lower frame rates.
  + Time.deltaTime is used throughout my project specifically for the movement of objects.
* **Interfaces**
  + Interfaces are a blueprint that defines methods a class must implement. It promotes loose coupling and allows polymorphism.
  + Interfaces become useful when dealing with classes that need to interact with classes of different types
* **Inheritance**
  + When creating a new class, that class can inherit properties and methods from an existing class. The sub class can add its own functionality or override inherited behaviour.
  + Inheritance is extremely useful for code organisation and reusability. When inheriting from the base class, you can avoid code duplication and create specialised versions of objects
* **Case pattern**
  + Case pattern is a structured approach to handling different conditions or scenarios in code, often using a switch statement or a series of if statements.
  + Case patterns can be useful for improving code readability, especially when dealing with multiple conditions.
* **Observer Pattern**
  + Observer Patterns are a design pattern that allows objects to observe changes in other objects and be told when those changes happen.
  + Observer Patterns are very useful for decoupling code and creating loosely coupled systems. It allows objects to react to events without needing to know the specifics of how those events occurred.
* **Polymorphism**
  + Polymorphism allows objects to respond to the same method call in different ways.
  + Polymorphism is very useful for creating flexible and generic code. It allows you to treat objects of different types in a similar way.
* **Communication between scripts/game objects**
  + Communication is essential for creating interactive game environments. Scripts can communicate through different ways such as public variables and methods or Unity’s event system.
  + Communication between scripts and game objects is the foundation of game logic and interactivity. It allows different parts of your games to interact and respond to each other’s actions.
  + Throughout my project, scripts interact and communicate with objects through references, such as referencing the target in my CameraMovement script, to allow the camera to lock on to an object, in this case the character.
* **Instantiation and Prefabs**
  + Instantiation is the process of creating a game object in the scene based on a prefab. Prefabs are reusable templates that store the properties of a game object.
  + Instantiation is extremely useful for performance optimisation and code maintainability. Prefabs allow you to create consistent and efficient copies of objects, reducing memory usage and simplifying code for managing common game elements
* **Magic Numbers** 
  + Magic Numbers are hardcoded numerical values directly embedded in code.
  + Magic Numbers can make code less readable, maintainable, and adaptable. I personally find it better to use variables or constants to represent these values.
* **Model Animation**
* **Self-made models and or animations**
* **Proper code placement**
  + Proper code placement refers to organizing your scripts in a logical and maintainable way. Structuring code based on functionality or placing scripts on appropriate game objects or components within the hierarchy for efficient code execution are two examples of how code can be easier to read and maintain.
  + Proper code placement helps the maintainability of code, making it well-organized makes it easier to understand, modify and debug. Performance can also improve by attaching scripts to relevant game objects, optimising script execution and avoiding unnecessary updates
* **Code repetition**
  + Code repetition is the practice of writing code that can be used in multiple places within your project or even across different projects. This involves creating generic and well-defined functions, components, and classes.
  + Code repetition also makes code easier to maintain, allowing you to change a single method, affecting all the places where it’s used. It also promotes consistency throughout your code, ensuring that similar functionalities behave the same way across different areas.
* **Feature 1**
* **Feature 2**
* **Feature 3**