**Functional Requirements:**

*Basic Movement*

* The player should be able to move in all four cardinal directions from a first-person perspective
* The player should be able to freely look around, but be unable to turn the camera upside down
* The player should have access to a basic jump ability that adheres to physics

*Grappling Hook Mechanics*

* The player should be able to engage and disengage the grappling hook using a button they have selected
* The grappling hook should connect to any solid platform or wall within a set range
* Upon attachment, the player should be tethered to the point selected, restricting their movement
* While attached, the player should be able to swing using their character’s weight as the mass of a pendulum

*Environmental Interactions*

* The player should be able to stand on, or connect their grappling hook to, any solid platform or wall in the level
* There should be certain types of object that the player must avoid touching
  + Touching any of these objects should return the player to the beginning of the level
* Levels should contain a bonus collectible item to encourage experimentation

*User Interface*

* The player’s UI should indicate if their current target point can be grappled to or not
* The UI should show the player’s time spent in the current level, as well as their number of restarts on this attempt
* The UI should show if the player has collected the bonus collectible item, either on the current attempt or any previous one
* The game should have a main menu where the player can change options, select a level, view leaderboards or exit the game
  + This menu should be presented to the user when they open the game
  + Level selection should indicate the player’s best time on that level, and whether they have collected its bonus collectible
  + The options submenu should provide the player options for camera sensitivity and inversion, as well as key bindings for various actions
  + The leaderboard should display multiple players’ names and their best recorded times for each level

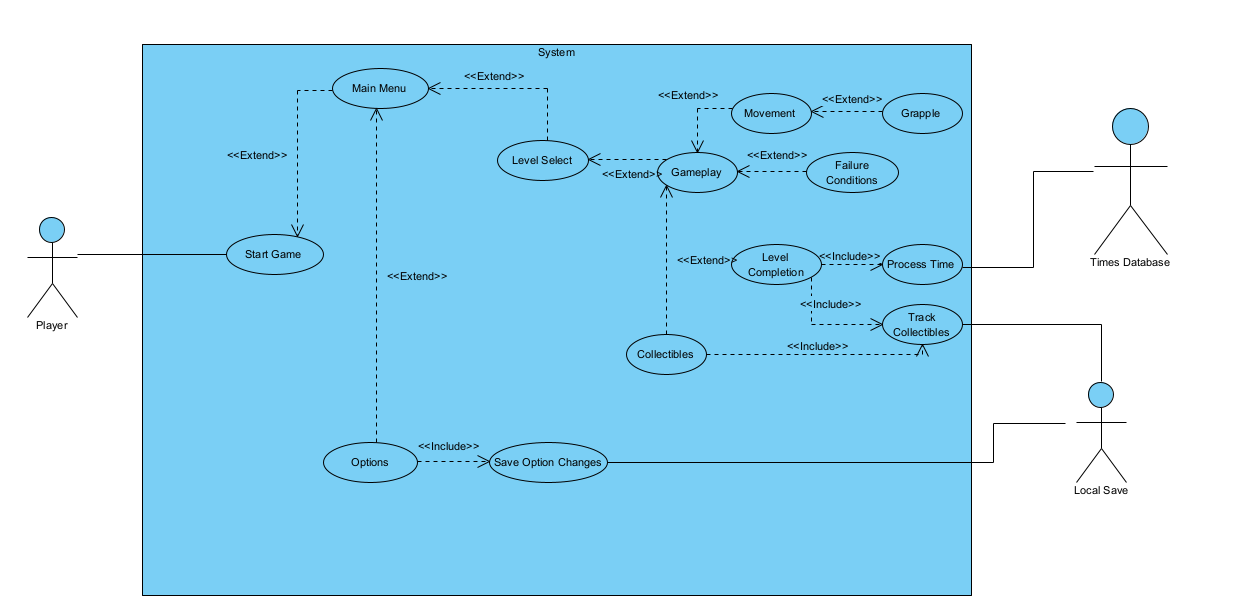
*Progression*

* Upon completing a level, the player should be shown their time and whether they collected the level’s collectible.
  + Players should be provided the option to continue to the next level or replay the previous one
* Player progress should be saved automatically upon completion of a level

**Non-Functional Requirements**

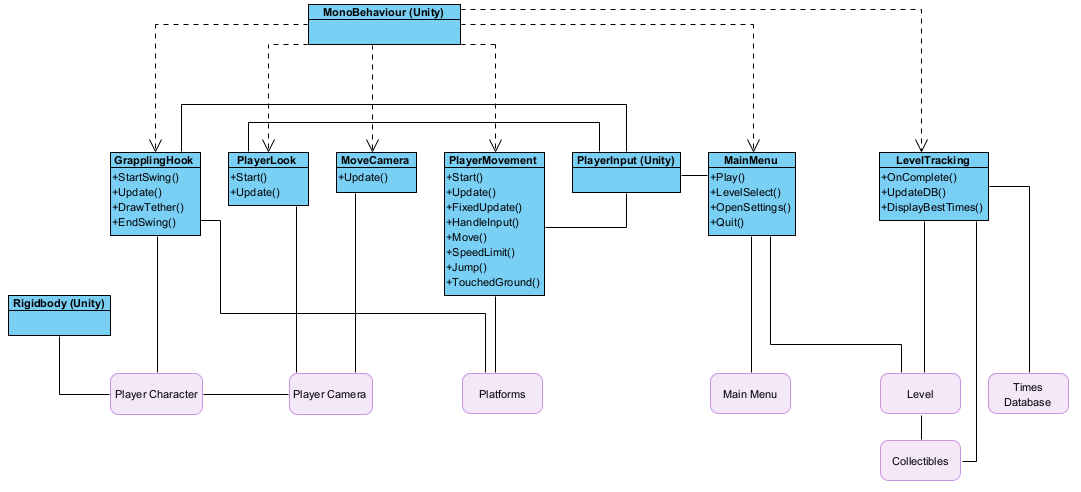
* During gameplay, the game should maintain a minimum frame rate of at least 60 frames per second on modern gaming-capable Windows platforms
  + The game should also maintain its frame rates consistently during gameplay, avoiding stuttering
* The game should be compatible with a range of Windows-based computers
* Load times should be minimised, taking no longer than 15 seconds to load a new level
* The game should be stable and avoid crashes to the desktop
* The user interface should be intuitive and easy to navigate
  + It should be legible at a range of screen resolutions
  + It should also scale appropriately with common display aspect ratios
* The game should be designed such that new levels or features can be easily added
* The player should be unable to directly tamper with the database of player times, or send it invalid entries
* The game should include a brief guide for how to play, to inform the player of the options they have during gameplay

**Diagrams**

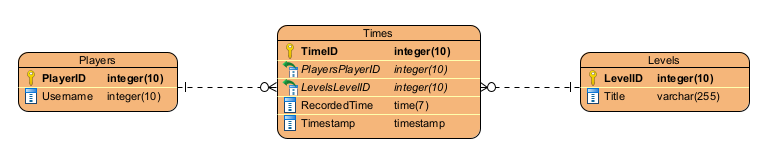
*Use Case Diagram:*

*Class Diagram*

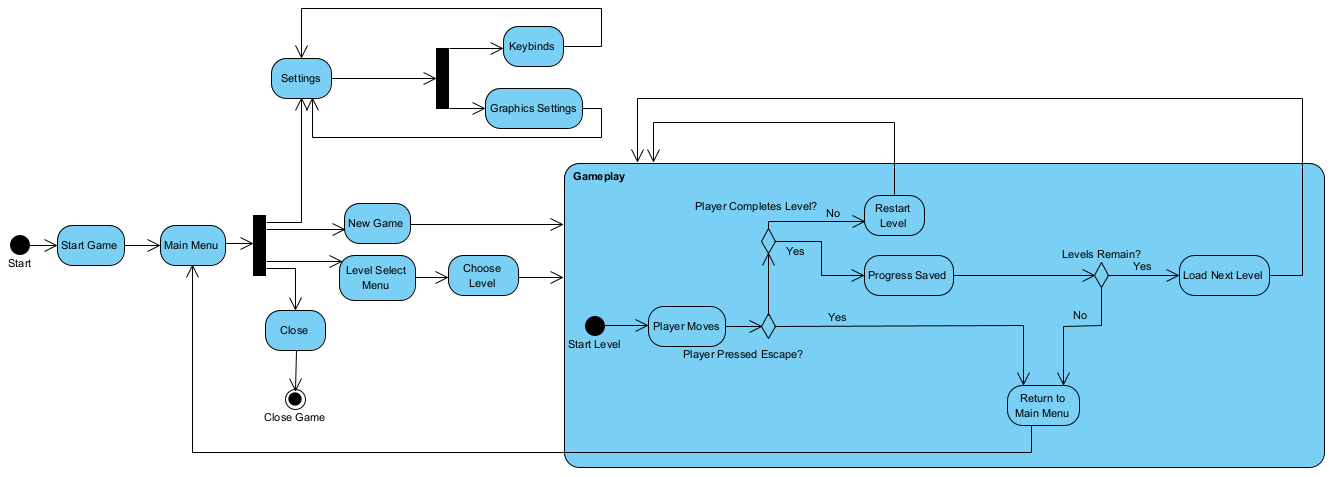
Note: As I am using some of Unity’s classes for my project, I will only be noting the most relevant parts here. There will likely be additions and changes made throughout this project, so this reflects my initial plans based on prototype development.

Given the flat structure, I feel some additional context is required to explain this diagram. MonoBehaviour is a class provided by Unity that allows for scripting in GameObjects. Scripts can be written that will be run when the associated GameObject is created or updated, among many other scenarios. GameObjects are the core building blocks of a game in Unity, for example characters, physical objects, lights or cameras. PlayerInput is auto-generated code that handles player input, based on selections made in Unity’s Editor. I have tried to capture the associations between my planned MonoBehaviour scripts and their associated objects, listed along the bottom of this diagram.

*Entity-Relationship Diagram*



*Activity Diagram*



*Sequence Diagram*