**Software Requirements and Design Document**

**For**

**Group 7**

Version 2.0

**Authors**:

Sarah Hall

Emily Schall

Justin Williams

Ahmed Alaoui

# Overview (5 points)

Grandeur is a 2D platforming puzzle video game. You begin the game in an unfamiliar castle with a large door with 8 individual locks. The goal of the game is to solve puzzles and other tasks to earn the 8 keys to unlock the big main door. Each member will be responsible for 2 “quests” which are either puzzles or achievements that the player must finish in order to beat the game.

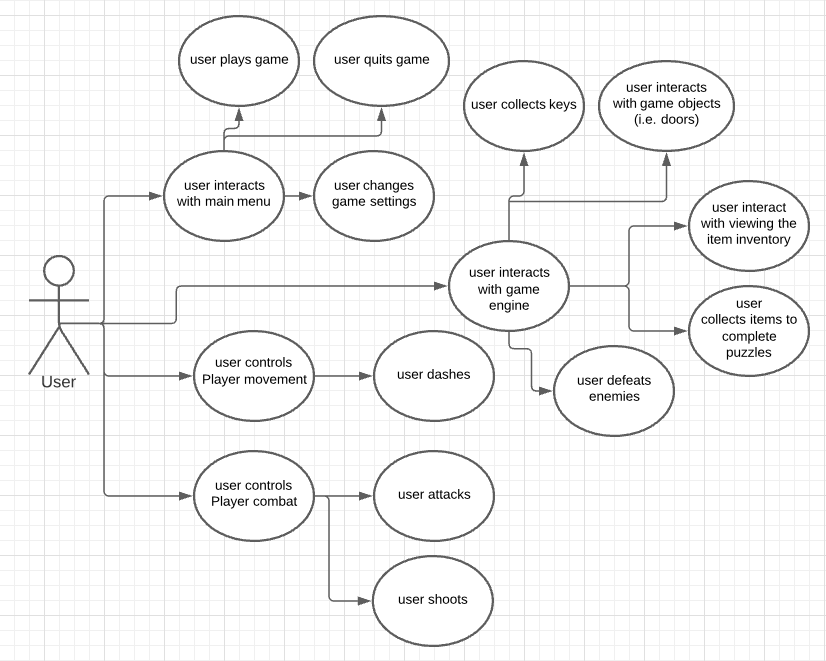
# Functional Requirements (10 points)

1. Player movement (walking, jumping, crouching, etc) - High
2. Player combat (attack, shooting, etc) - Medium
3. Player animations (idle, walking, jumping, etc) - Low
4. Enemy animations (idle, walking, jumping, etc) - Low
5. Environment animations - Low
6. Enemy combat (attack, shooting, etc) - Medium
7. Tracking Camera - High
8. Hub room of the main door and side doors - Medium
9. storage/item system - Medium
10. settings and save menu - High
11. “quests” screen that shows which keys you have and what is remaining - Medium
12. Map of the castle, showing only visited rooms/dungeons - Low

# Non-functional Requirements (10 points)

1. smooth movement, smooth abilities
2. enjoyable gameplay
3. able to save data and retrieve saved game data
4. user is not able to modify save data
5. reasonable response time in game

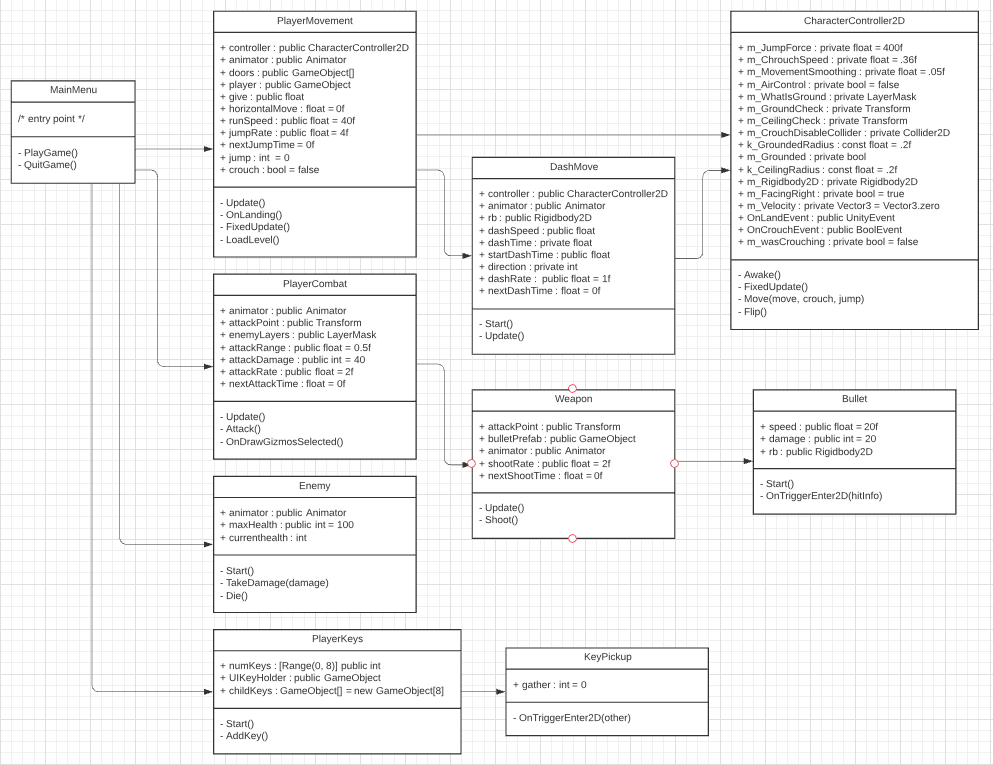
# Use Case Diagram (10 points)



When a user begins to play our game they can interact with the game in four main ways.

1. The User can interact with the main menu which will allow them to play the game, quit the game, or change game settings such as volume.
2. When playing the game, the User can interact with the game engine to perform tasks that progress the game. This includes collecting keys/items, defeating enemies, interacting with game objects such as doors, viewing the item inventory, and completing puzzles
3. When playing the game, the User can control the basic Player movements (run left/right, jump, crouch). These basic movement controls are enhanced by extra functions such as the dash ability
4. When playing the game, the User can use combat functions to defeat enemies. The User can attack using the Player’s sword and shoot using the Player’s abilities.

# Class Diagram and/or Sequence Diagrams (15 points)



This is an updated class/sequence diagram that reflects the new functionalities added this iteration. A more detailed class/sequence diagram to be included in the next iteration as there is more code that will display class interactions and the interaction between objects or functions.

The general overview of the system (or assumed to be) as follows:

* A UIManager script that controls and saves all user info, gameplay, settings, keys found, inventory
* Gameplay script that controls player movement
* Gameplay script that controls player combat
* Gameplay script that controls item pickup/interaction
* Enemy script the controls general enemy movements
* An Animator that controls animation for the player, enemies, and environment
* Camera that tracks player movement
* Various quest scripts to control each quest

# Operating Environment (5 points)

This video game is made using C# and Unity and will be ported to work on “PC, Mac, Linux, and WebGL”, so any common computer regardless of the OS.

# Assumptions and Dependencies (5 points)

We are currently using free or previously purchased game assets but if time allows, we would like to create our own assets. We plan on using music developed by a local musician and friend, or just using free music. We are also using free unity packages for some in game functions such as Cinemachine for a tracking camera.

A combat system has been implemented, but is not fully fleshed out the combat system yet, so that may be scrapped depending on time and if we want to only focus on puzzles and platforms. Ideally some levels will be puzzle focused, some will be combat focused, and some may have a mixture of the two.