**Software Requirements and Design Document**

**For**

**Group 7**

Version 3.0

**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, defeat enemies and complete multiple platforming tasks to earn the 8 keys. 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.

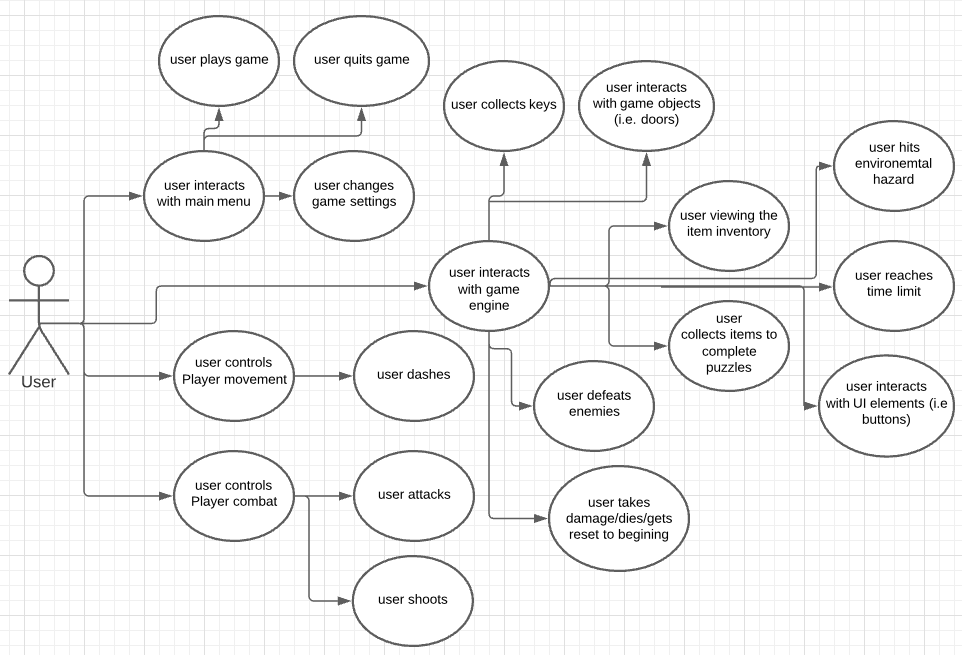
# Functional Requirements (10 points)

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. Main Menu - 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

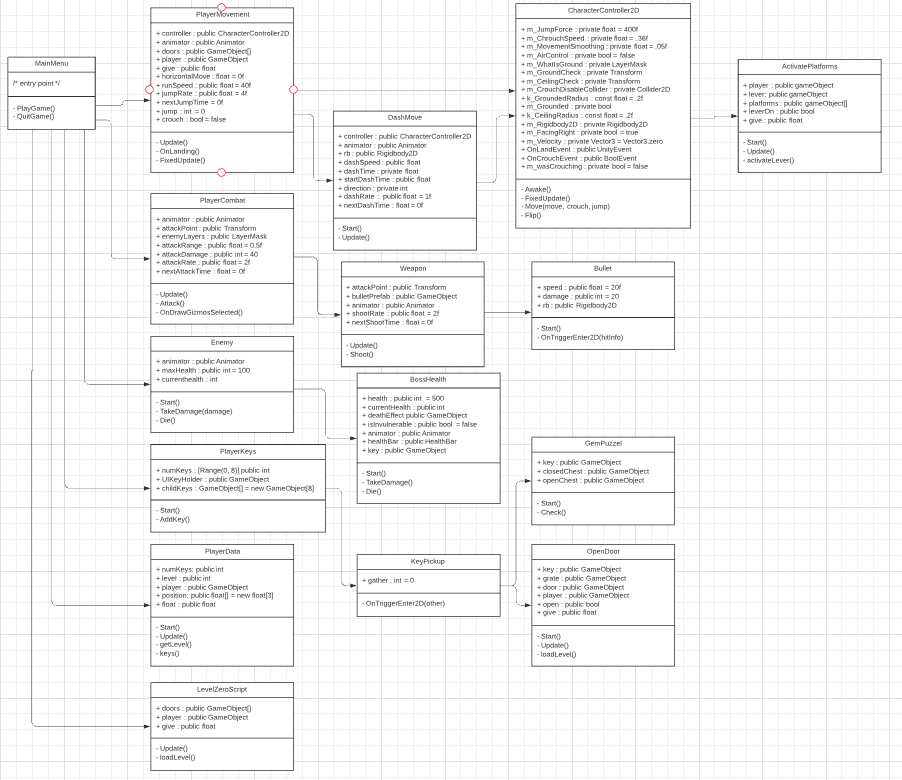
# Use Case Diagram (10 points)



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 interacting with UI elements, viewing the item inventory, and completing puzzle
3. When playing the game the user can also be limited by interactions with the game engine. This includes environmental hazards, projectiles, and time limits.
4. 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
5. 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.There are other functions to be included in this diagram however the software used to create the diagram (lucid.app/lucidchart/) does not allow the use of more assets without a paid subscription.

Here is what should also be included in the diagram:

* BossWeapon, Boss\_Enrage, Boss\_Run, and Boss are all classes that would be attached to the Enemy class on the diagram as they operate within that functionality.
* PlayerHealth class would be attached to the MainMenu class as it is a functionality that is established from the beginning.
* HazardDamage, projectile, projectileDamage, Spike, Moving Platform, Moving Platform Transporter and DeploySwords would be attached to the MainMenu class as they are functionalities that are established on their own from the start of the game.
* LevelSeven and Level8Script would be attached to the MainMenu class as they are functionalities that are established on their own from the start of the game.
* ShowInsructions would be connected to CharacterController2D as its functionality depends on the player’s ability to move.

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

* A UIManager script that controls all user info, gameplay, settings, keys found
* Gameplay script that controls player movement
* Gameplay script that controls player combat
* Gameplay script that controls item pickup/interaction
* Gameplay script that controls object and UI interaction
* Enemy script that controls general enemy movements
* Hazard script that controls interactions with environments
* 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)

The game uses free or previously purchased game assets. Time did not permit us to be able to create our own assets or music for the game. As a result, many of the games environments do not blend together well, but they add diversity and style to the game build.We also used free unity packages for some in game functions such as Cinemachine for a tracking camera.

A combat system has been implemented on specific levels through boss fights. The combat is relatively simple but has a few nuances that allow for creativity in gameplay such as a super-double jump. There are also puzzle systems implemented through the UI and key pick-up system that players will depend on in order to proceed through the level. Key tracking across levels was not able to be implemented so the player must assume that the game will not remember if you have beaten a level.