**Software Requirements and Design Document**

**For**

**Group 2**

Version 1.2

**Authors**:

Zachary Ortiz

Rafe Ewert

Joe Sahl

# Overview (5 points)

The game is a recreation of Super Mario Bros from the NES, and is being built in Godot, a game engine that supports 2D platformers. It is being written in Godot’s own language, GDScript, but that may be subject to change at a later date. The game will have three levels: Overworld, Underground, and Castle. Each level will have its respective difficulties, such as enemy count, obstacle count, and item count. (Z)

# Functional Requirements (10 points)

1. A working player control scheme so that, based on keys pressed, Mario will move how the player controlling him wants him to move (High) (Z)
2. A functioning level system with transitions and proper collision (High) (Z)
3. Working enemies and obstacles (Medium) (Z)
4. Accurate representation of character and enemy animation and sound design (Medium) (J)

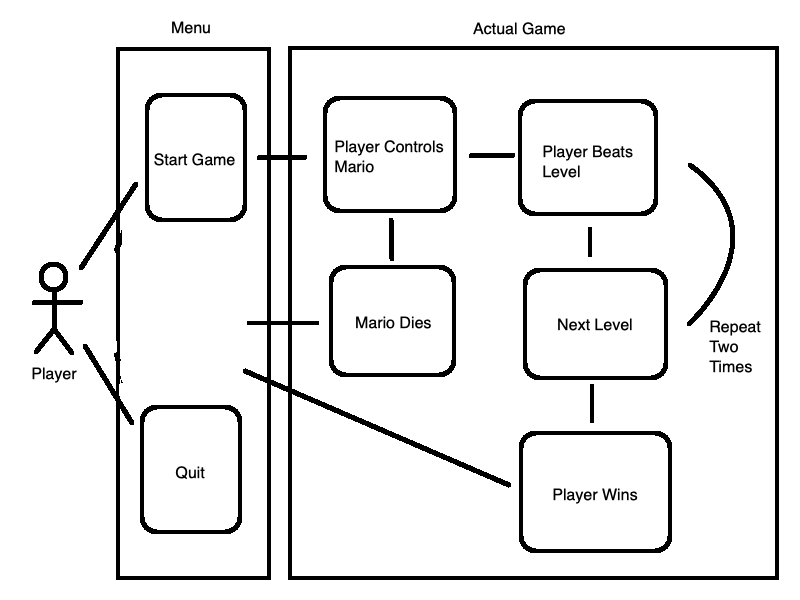
*List the* ***functional requirements*** *in sentences identified by numbers and for each requirement state if it is of high, medium, or low priority. Each functional requirement is something that the system shall do. Include all the details required such that there can be no misinterpretations of the requirements when read. Be very specific about what the system needs to do (not how, just what). You may provide a brief design rationale for any requirement which you feel requires explanation for how and/or why the requirement was derived.*

# Non-functional Requirements (10 points)

1. Mario being capable of death, and upon death, a “game over” screen will appear. (Z)
2. Mario will retain whatever power ups he received in the previous level when transitioning to the next. (Z)
3. Enemies will persist even while off-screen to discourage the player from moving past enemies instead of eliminating them. (J)
4. Software should be able to maintain consistent FPS while the enemies and objects are all loaded. (R)

*List the* ***non-functional requirements*** *of the system (any requirement referring to a property of the system, such as security, safety, software quality, performance, reliability, etc.) You may provide a brief rationale for any requirement which you feel requires explanation as to how and/or why the requirement was derived.*

# Use Case Diagram (10 points)



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

*This section presents a high-level overview of the anticipated system architecture using a* ***class******diagram*** *and/or* ***sequence diagrams****.*

*If the main* ***paradigm*** *used in your project is* ***Object Oriented*** *(i.e., you have classes or something that acts similar to classes in your system), then draw the* ***Class Diagram******of the entire system and Sequence Diagrams for the three (3) most important use cases in your system.***

*If the main* ***paradigm*** *in your system is* ***not Object Oriented*** *(i.e., you* ***do not*** *have classes**or anything similar to classes in your system) then only draw* ***Sequence Diagrams****,* ***but for all the use cases of your system.*** *In this case, we will use a modified version of Sequence Diagrams, where instead of objects, the lifelines will represent the functions in the system involved in the action sequence.*

***Class Diagrams*** *show the* ***fundamental objects/classes*** *that must be modeled with the system to satisfy its requirements and* ***the relationships*** *between them. Each class rectangle on the diagram* ***must also include the attributes and the methods of the class*** *(they can be refined between increments). All the* ***relationships between classes and their multiplicity*** *must be shown on the class diagram.*

*A* ***Sequence Diagram*** *simply depicts* ***interaction******between objects*** *(or* ***functions -*** *in our case - for non-OOP systems) in a sequential order, i.e. the order in which these interactions take place. Sequence diagrams describe how and in what order the objects in a system function.*

# Operating Environment (5 points)

The game will be playable on any kind of computer, be it desktop, laptop, Windows, or Mac. It will not run on any other kind of platform. (Z)

# Assumptions and Dependencies (5 points)

1. GDScript can be easily translated in to C++. (Z)
2. Assets such as sprite maps and sheets are taken from The Spriters Resource website. (Z)

*List any assumed factors (as opposed to known facts) that could affect the requirements stated in this document. These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints. The project could be affected if these assumptions are incorrect, are not shared, or change. Also identify any dependencies the project has on external factors, such as software components that you intend to reuse from another project.*