Software Requirements and Design Document

For

Group <30>

Version 1.0

Authors:
Griffin Guthrie
Javier Quiroz
Wesley Yawn
Chris Cocose

1. Overview (5 points)

Give a general overview of the system in 1-2 paragraphs (similar to the one in the project proposal).

Our project is a 2D combat/puzzle platformer similar in design to Metroid. The player plays as a robot traversing a mechanical wasteland gathering several attachments to help progress each level. There will be a few levels and each will introduce a new attachment for the player to use and will help them through the level.

2. Functional Requirements (10 points)

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.*

1. Creating a Player control script so that a player uses A and D to move left and right and Space to jump. (HIGH)
2. Designing a playable level. Must have a level that a player can play or there is no game. (HIGH)
3. Having Scripts for Physics and Collision. This is so that a player can collide with objects instead of falling through them and determining how a player moves such as speed when falling. (MEDIUM)
4. Camera Follow System. This is essentially developing the program to follow the player character as they move across the screen. (MEDIUM)
5. Attachments for the player character. Players will be able to pick up and equip items that allow for a user to complete the level such as a grappling hook for crossing large gaps.(MEDIUM)
6. Enemy AI. This is so that enemies will actively attack the player according to set patterns. (LOW)
7. Pausing and Menus that allow for a player to stop the game and exit the game if they so desire. (LOW)
8. Player Sprite Animations. This is just having the player character have animations when walking, jumping or using an item. (LOW)

3. Non-functional Requirements (10 points)

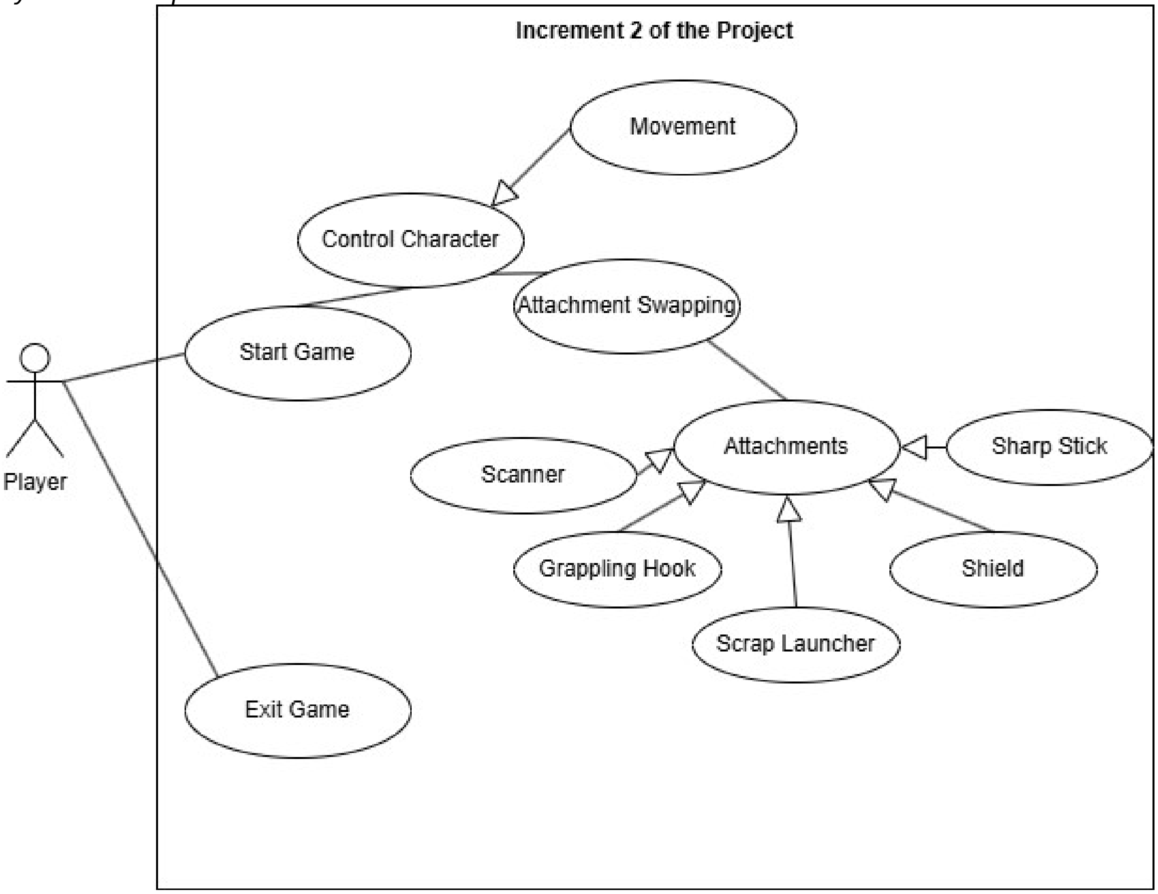
List the ***non-functional requirements*** *of the system (any requirement referring to a property of the system, such as security, safety, software quality, performance, reliability, etc.) You may provide a brief rationale for any requirement which you feel requires explanation as to how and/or why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

- Transition between levels. When one level is beat the program must move to the next level smoothly.
- Player Sprite animations look good as in the animations are smooth and don’t look like the game is lagging or losing frame rate.
- Useful UI/Menus for the player that don’t take up a lot of space on the game screen and have what is necessary for those menus.
- Easy to read text and icons that users can easily see and comprehend.
- Audio cues for items and the environment that the users can understand and tell what is happening in the game.

4. Use Case Diagram (10 points)

This section presents the ***use case diagram*** *and the* ***textual descriptions*** *of the use cases for the system under development. The use case diagram should contain all the use cases and*

relationships between them needed to describe the functionality to be developed. If you discover new use cases between two increments, update the diagram for your future increments.
Textual descriptions of use cases*: For the first increment, the textual descriptions for the use cases are not required. However, the textual descriptions for all use cases discovered for your system are required for the second and third iterations.*



Start Game is pretty self-explanatory it starts the program.
Exit Game exits the program.
Control Character is a generic description for all the things the player character will be able to do.
Movement is a subset of control character and specifically deals with how the player will move such as sprinting, moving left and right, and jumping.
Attachment Swapping is the ability to swap between any of the attachments.
Attachments are all the different things a player can do for unique movement (such as grappling) or to find items or hitting enemies.
Scanner finds items on the map.
Grappling Hook pulls the player character towards something and lets them hang off a rope.
Scrap Launcher is able to fire scrap at an enemy.
Shield is to block any incoming damage.
Sharp Stick is used for melee and damaging enemies.

5. 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****.*

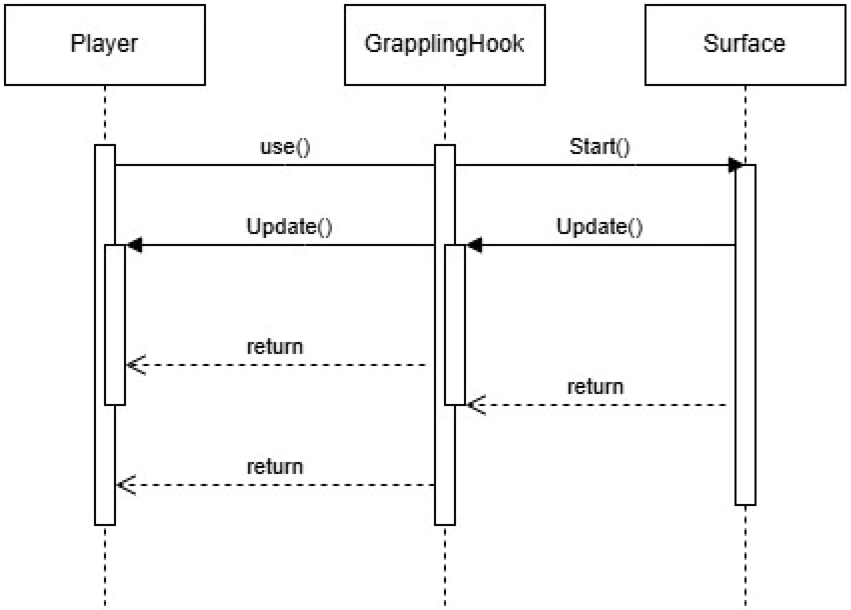
If the main ***paradigm*** *used in your project is* ***Object Oriented*** *(i.e., you have classes or something that acts similar to classes in your system), then draw the* ***Class Diagram of the***

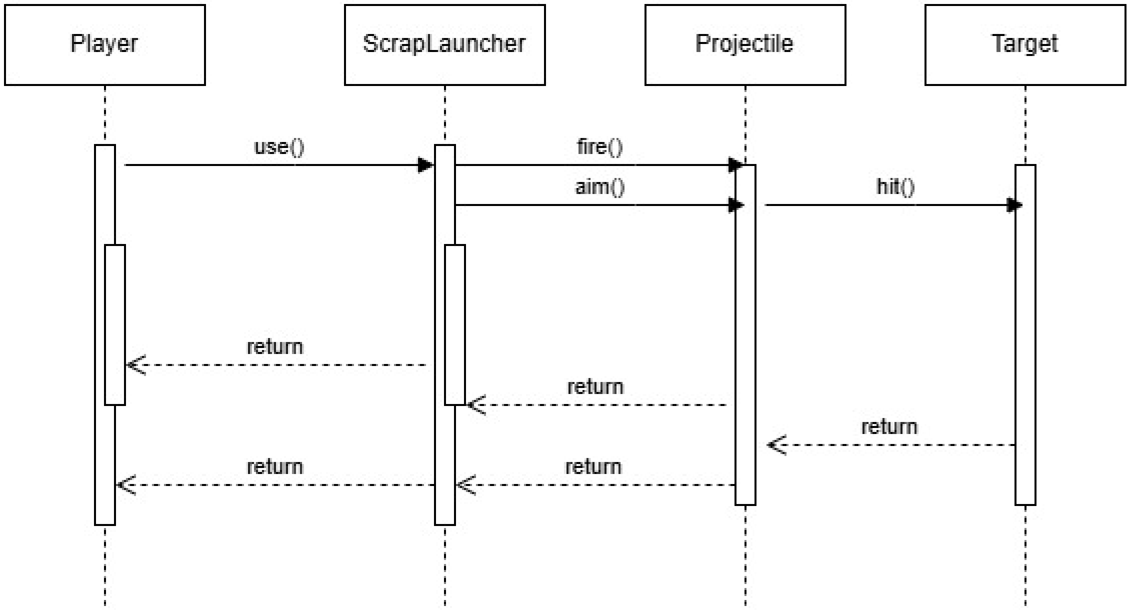
entire system and Sequence Diagrams for the three (3) most important use cases in your system.

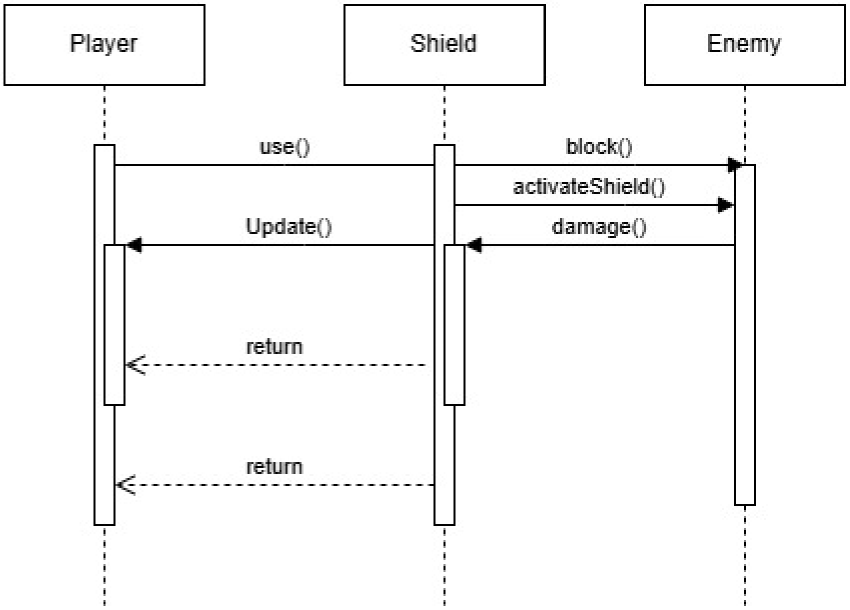
*If the main **paradigm** in your system is **not Object Oriented** (i.e., you **do not** have classes or anything similar to classes in your system) then only draw **Sequence Diagrams**, **but for all the use cases of your system.** In this case, we will use a modified version of Sequence Diagrams,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


Player
GrapplingHook
Surface
use()
Start()
Update()
Update()
return
return
return


Player
ScrapLauncher
Projectile
Target
use()
fire()
aim()
hit()
return
return
return
return
return


Player
Shield
Enemy
use()
block()
activateShield()
Update()
damage()
return
return

i

6. Operating Environment (5 points)

Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist.

The hardware platform will be on computers using Windows 10/11 as that is what we currently use when developing in Unity. Other than that while in development it will need Unity to run until it is turned into an executable file. We are currently using Unity 6000.0.33f1 and our program has worked with newer versions but we recommend Unity 6000.0.33f1.

7. Assumptions and Dependencies (5 points)

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.

- An assumed factor is asset availability as we are trying to develop all our own sprites and are using the assets given in Unity that are free. This could slow us down since it takes time to develop custom sprites.
- An assumption we have is Unity being version compatible and being able to use our program with other versions of Unity. Which may result in issues if Unity introduces major changes with their next update.
- Another assumption is Unity’s physics and animation systems being sufficient for our project.
- A dependency is that we rely on Unity as our platform for development. If unity has major updates we would need to rework our project significantly.