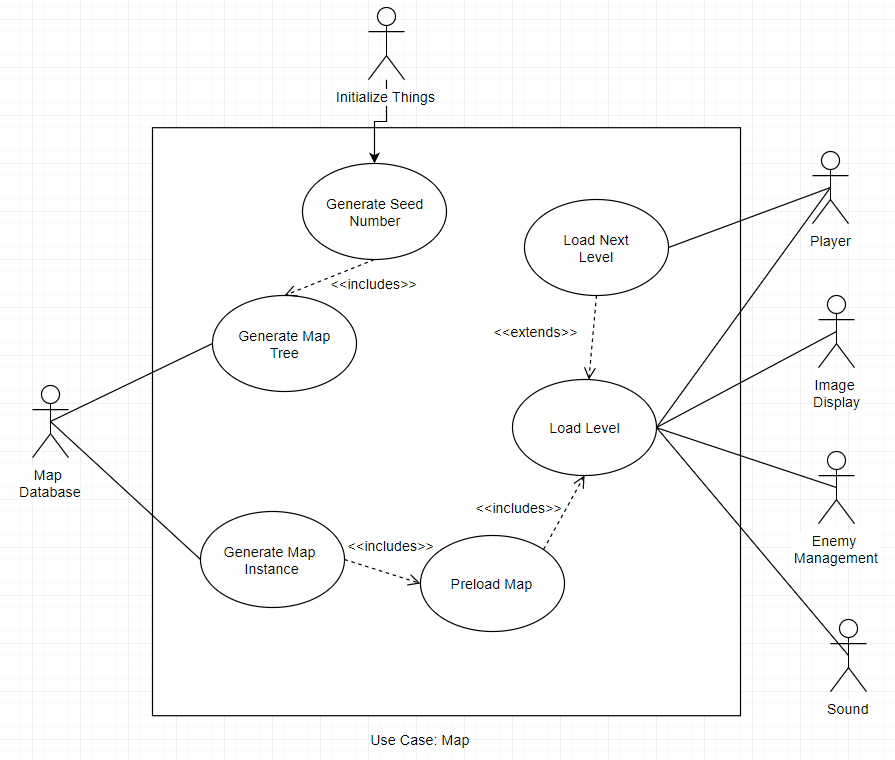
Name: James Todd Mark \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/50

## Brief introduction \_\_/3

I’ll be creating the maps for the levels in our game. To do so, I intend to take a signal from the UI when the game begins, which will trigger the generation of a seed number. This will then be used in conjunction with the map database to generate the combination of rooms that makes up each of the levels, according to a specific set of rules. The maps will be preloaded into memory, and then passed on to the various systems that use them.

## Use case diagram with scenario \_\_14

### Use Case Diagram



### Scenarios

**Name:** Generate Seed Number

**Summary:** The Initialize Things actor initiates game start, which triggers the map to generate a random seed number

**Actors:** Initialize Things

**Preconditions:** Game has been started

**Basic Sequence:**

**Step 1:** Accept flag from Initialize Things

**Step 2:** Call Rand() function

**Step 3:** Send result to Generate Map Tree process

**Exceptions:** None

**Post Conditions:** Random Value has been generated and sent to Generate Map Tree process

**Priority:** 2

**ID: ADD LATER**

**Name:** Generate Map Tree

**Summary:** Generate Seed Number initiates the Generate Map tree function, and returns a struct that contains the map and connections based on a set of rules, it then sends this information to the map database.

**Actors:** Map Database

**Preconditions:** Random seed has been generated.

**Basic Sequence:**

**Step 1:** Accept seed from Generate Seed Number

**Step 2:** Run seed through state machine

**Step 2a:** The state machine will bind rooms together

**Step 3:** Take output from state machine, in tree form, and retrieve related information from map database.

**Exceptions:** None

**Post Conditions:** Seed has been processed into tree form

**Priority:** 2

**ID:**

**Name:** Generate Map Instance

**Summary:** Generate Map Instance links the rooms together to form the individual levels and map. Afterword, it sends the map information to the preload map function.

**Actors:** Map Database

**Preconditions:** The map tree has been generated, and the map database has send back the relevant room information.

**Basic Sequence:**

**Step 1:** Accept information from map database relevant to building levels.

**Step 2:** Link the levels together using Unreal’s Level Streaming function, while traversing the tree

**Step 3:** Send the linked level to the Preload Map function

**Exceptions:** None

**Post Conditions:**  Map has been linked into one structure.

**Priority:** 1

**ID:**

**Name:** Preload Map

**Summary:** Adds objects into the level, and loads the level into main memory

**Actors:** None

**Preconditions:** Map has been generated.

**Basic Sequence:**

**Step 1:** Accept the map information from Generate Map Instance

**Step 2:** Links objects into each room

**Step 3:** Call Unreal’s LoadLevel function

**Step 4:** Send level to Load Level process

**Exceptions:** None

**Post Conditions:** Map is loaded into memory, with all the relevant non-enemy, non-player objects linked into it.

**ID:**

**Name:** Load Level

**Summary:** Loads player and enemies into level, as well as sound

**Actors:** Player, Enemy Management, Image Display, and Sound

**Preconditions:** Rest of level has been linked and loaded into memory

**Basic Sequence:**

**Step 1:** Accept Map information from the Preload Map process

**Step 2:** Communicate with Enemy actor to get objects for enemies, and link their instances to the proper locations according to each room’s data.

**Step 3:** Do the same thing with the player actor

**Step 4:** Send level information to Sound actor to start background music

**Step 5:** Send the completed level to the Image Display actor

**Exceptions:** The start signal from this function can come from Load Next Level as well. This loads the next level into memory rather than the first one.

**Post Conditions:** The next level to be played is loaded with

**ID:**

**Name:** Load Next Level

**Summary:** Calls the Load Next Level process when the player actor has completed the current level (when player collides with the goal)

**Actors:** Player

**Preconditions:** Player has completed level, signal passed from player actor

**Basic Sequence:**

**Step 1:** Receive signal from player actor

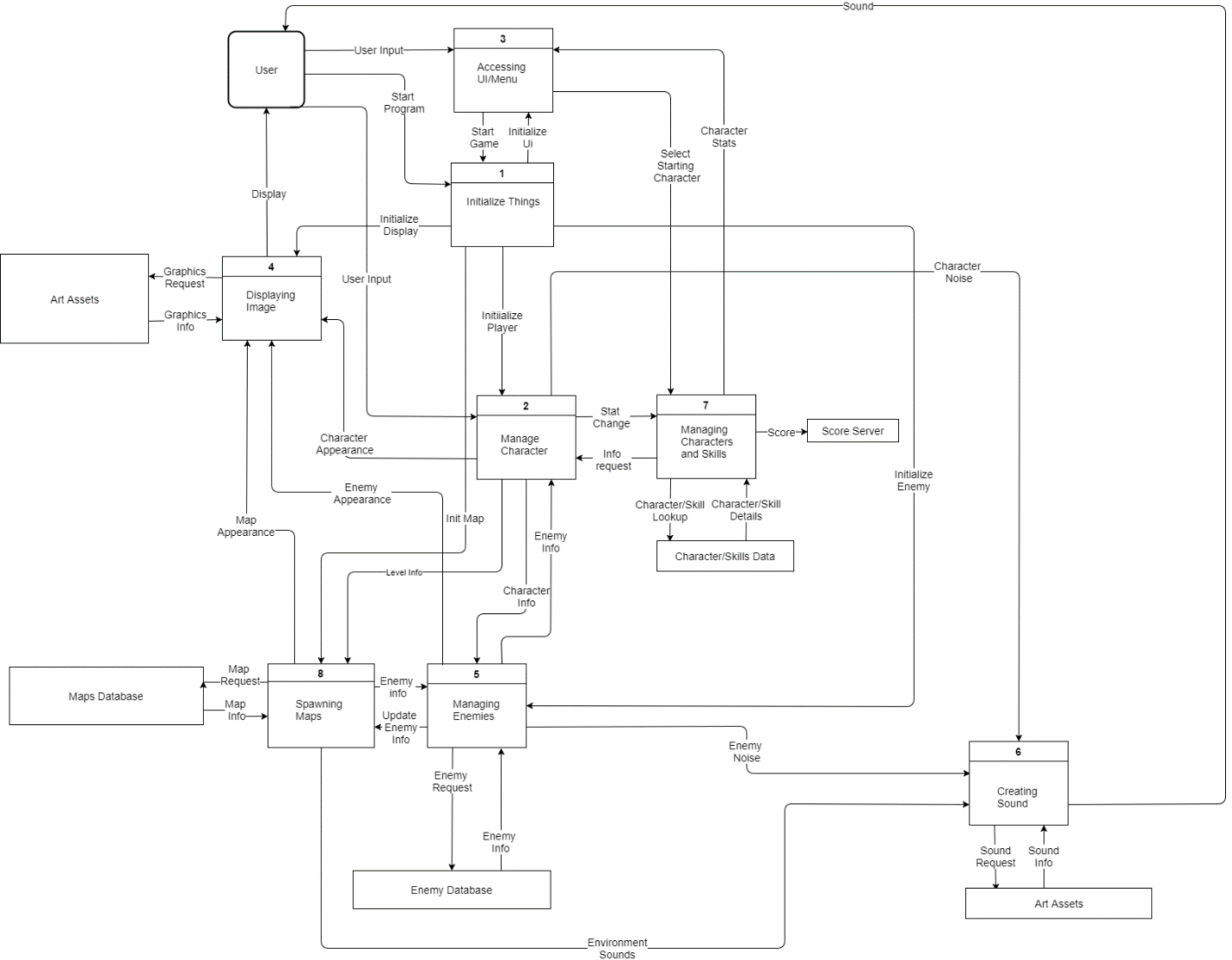
**Step 2:** Send signal and information to Load Level process

**Exceptions:**  None

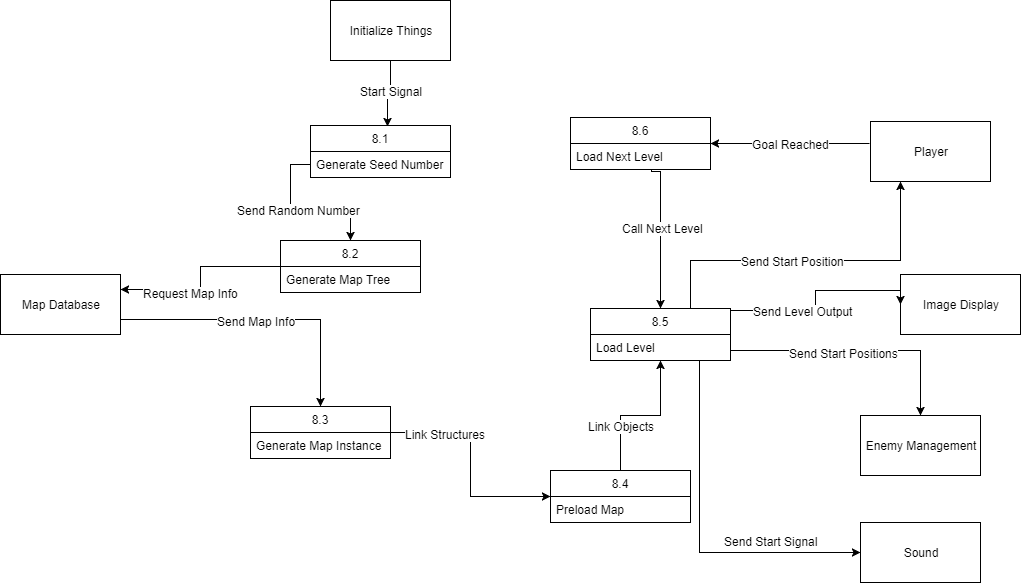
**Post Conditions:** The Load Level process is called with the next set of arguments

**ID:**

## Data Flow diagram(s) from Level 0 to process description for your feature \_\_\_\_\_\_\_14



### Data Flow Diagrams



### Process Descriptions

Generate Seed Number: This function takes will be triggered by a call from the “Initialize Things” process. It will call a rand() function, and return this string to the Generate Map Tree function. This is to work as a seed value to stochastically generate the tree that will represent the way the rooms and levels are connected.

Generate Map Tree: This function takes input from the Generate Seed Number function. It will then run this string through a cellular automaton to process the string into a tree that represents the way the rooms and levels are connected. This will allow rules to be implemented as to how the map is generated. The completed tree will then be sent to the map database to request the listed rooms, which will then be sent onto the Generate Map Instance function.

Generate Map Instance: This function will take input from the Generate Map Tree function. It will then take the map data and tree from the map tree function to link the rooms together into levels within the map. This new structure will then be sent to the Preload Map function.

Preload Map: This function will take input from the Generate Map Instance function, and will then link the called objects from each room into the map. It will then pass this on to the Load Level function.

Load Level: This function will take input from the Preload Map function. It will then send this information everywhere it needs to go—to the Enemy, player, sound, and display processes.

Load Next Level: This function will take input from the player process, when the player collides with the GoalStateObject(). It will then call Load Level, with the input being the next level after the one the player has just completed.

## Acceptance Tests \_\_\_\_\_\_\_\_9

Generate Seed Number:

Run feature 1000 times sending output to a file.

The output file will have the following characteristics:

It will consist of letters and digits, and be 50 characters long.

Each digit will appear within 3 standard deviations of each other

No substring may appear more than 3 times, for substrings of len(10)+

Generate Map Tree:

Run feature 1000 times, sending output to a file, with input from previous test.

The output file will have the following characteristics:

It will consist of 1000 trees, with 4 levels each, with rooms in each level equal to at least 15 rooms.

No doors are present in the level system that lead nowhere.

A goal is present in each level.

Generate Map Instance:

This feature will be run 50 times, with output being sent to the user interface. Each level will be walked through to ensure that it is playable.

The output levels will have the same features as listed above, but will all display as intended.

Preload Map:

This feature will be run 50 times, with output being sent to the Load Level UI in Unreal.

The output will all be formatted as a single package, able to be read at once without errors.

Load Level:

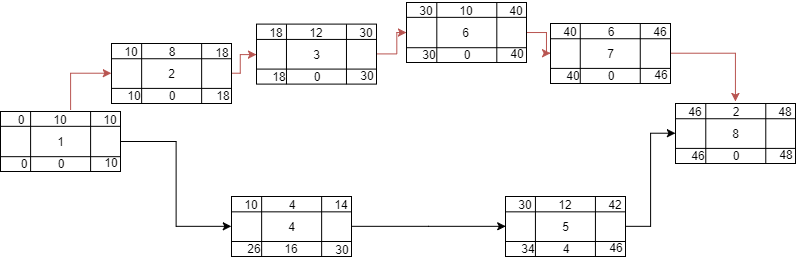
This feature will be run 50 times, with output being sent to each of its attached processes, to make sure that data is formatted correctly for the input of each of them, depending on the inputs required by said feature. For example, the signal to Sound will be made to be a trigger containing the level number, while the output to player will be a start position.

## Timeline \_\_\_\_\_\_\_\_\_/10

### Work items

|  |  |  |
| --- | --- | --- |
| Task | Duration (hours) | Predecessor Task(s) |
| 1. Requirements Collection | 10 | - |
| 2. Level Design | 8 | 1 |
| 3. Level Implementation | 12 | 2 |
| 4. Learning Unreal | 4 | 1 |
| 5. User Documentation | 12 | 4 |
| 6. Programming | 10 | 4 |
| 7. Testing | 6 | 6 |
| 8. Installation | 2 | 5, 7 |

### Pert diagram



### Gantt timeline

