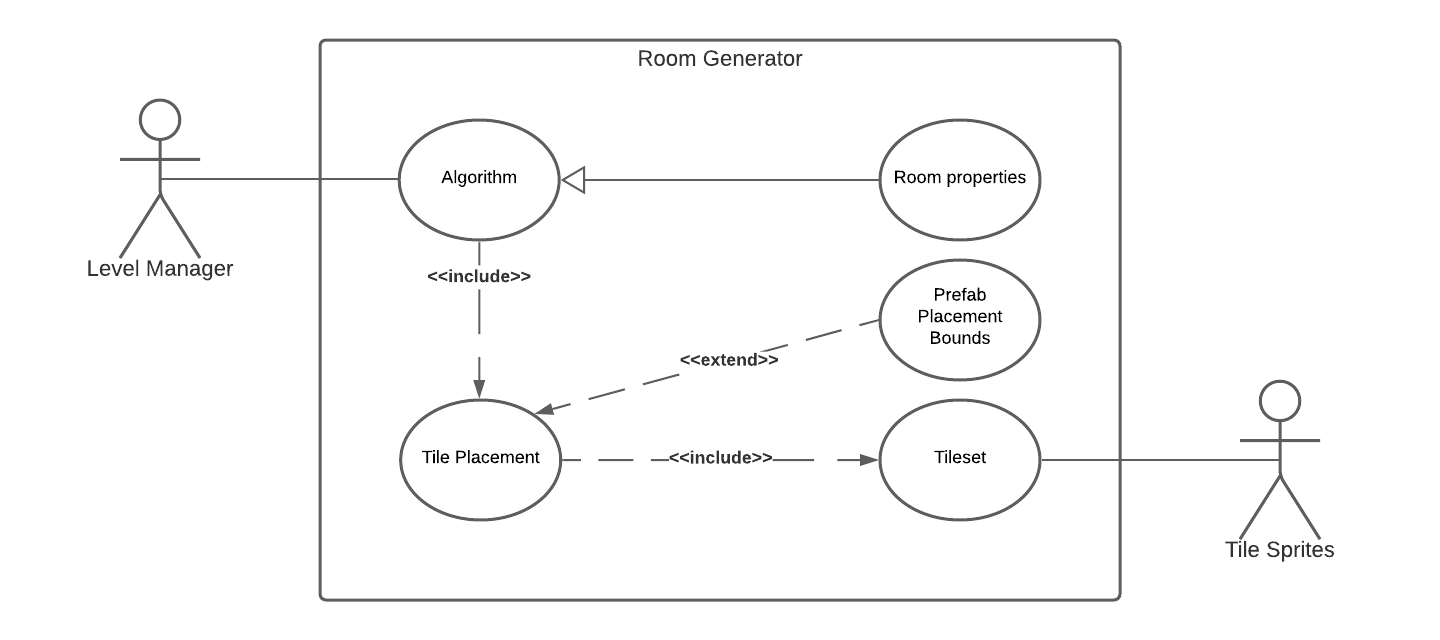
Name **Ezequiel Romero**  Mark \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/50

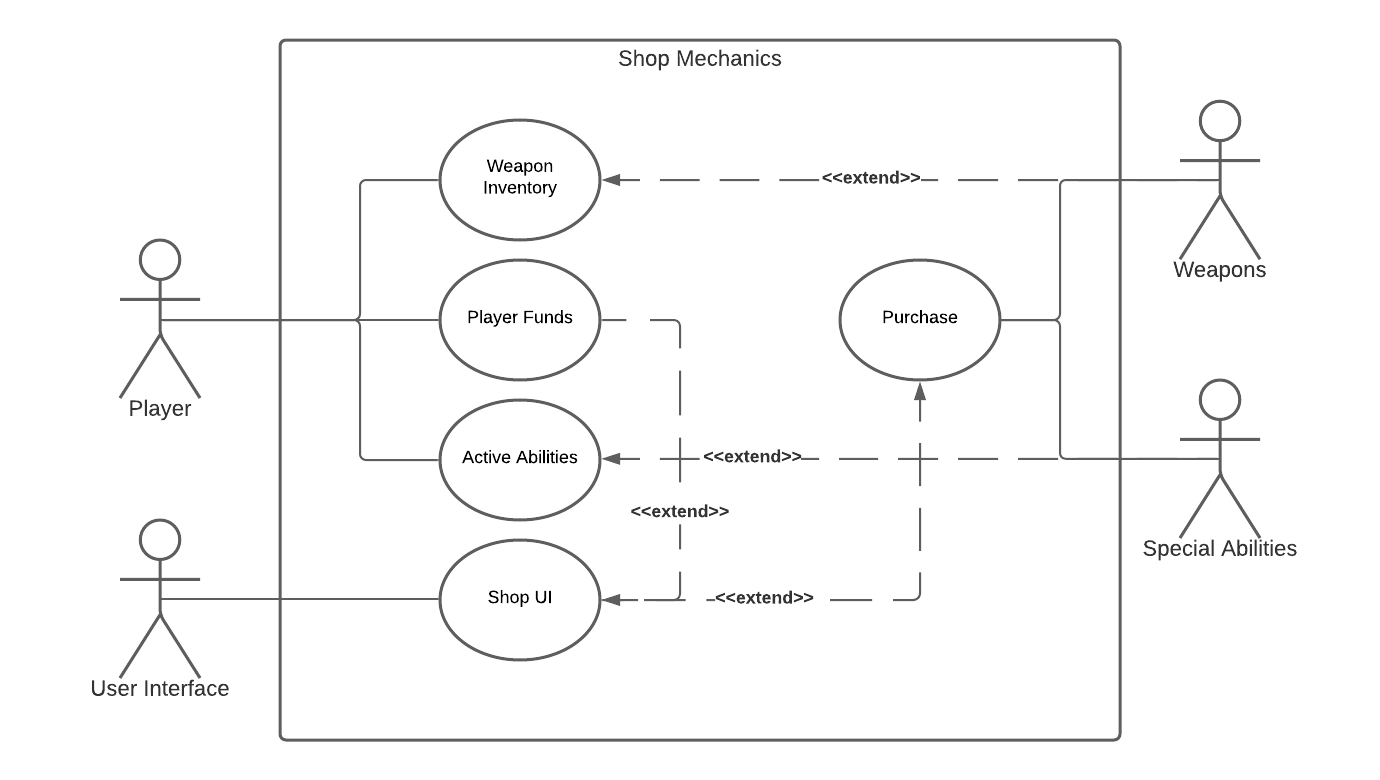
## Brief introduction \_\_/3

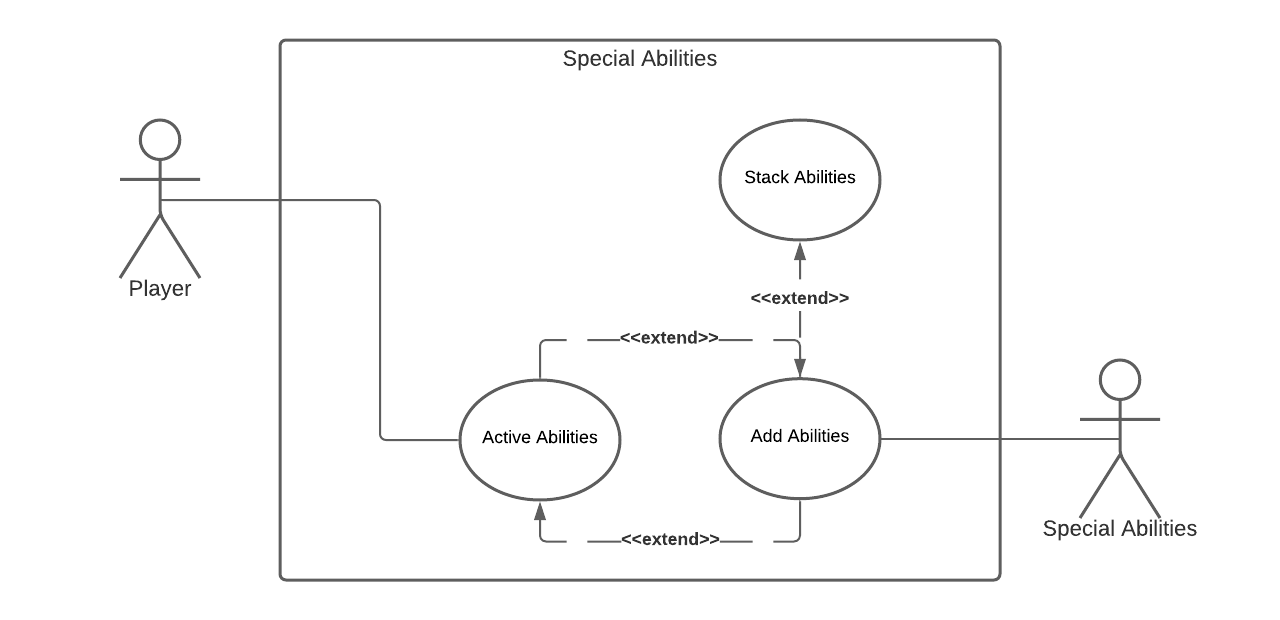
I will be implementing **procedural room generation** into our game. This will be done, first by generating an algorithm for determining rooms and paths in a simple array. Then, I will be able to convert this array into an actual playable room with custom tile sets. I will also be working on **Shop Mechanics**, and **Special Abilities**. The shops around the play areas will most likely be futuristic vending machines that sell weapons, health packs, and maybe special abilities for purchase. The special abilities will alter the playstyle of the player, allowing the player to specialize in whatever mechanic they choose. The special ability will level up as they continue to use said ability. Multiclassing will also be included.

## Use case diagram with scenario \_\_14

### Use Case Diagrams







### Scenarios

**Name:** Algorithm

**Summary:** The accountant uses the machine to calculate the sum of two numbers.

**Actors:** Level Manager.

**Preconditions:** Procedural Room Generation is called from Level Manager, if the room is not planned to be designed.

**Basic sequence:**

**Step 1:** Access room properties

**Step 2:** Perform Algorithm based on room properties

**Step 3:** Place result into array

**Exceptions:**

**Step 1:** If the algorithm is unable to calculate a playable area based on the room properties, it will decrease values until it is able to produce a play area.

**Post conditions:** Calculated array goes to

**Priority:** 1\*

**ID:** RG1

**Name:** Room Properties

**Summary:** The Algorithm takes in these settings

**Actors:** Level Manager.

**Preconditions:** No preconditions are necessary for this use case

**Basic sequence:**

**Step 1:** Get input settings from editor

**Step 2:** Apply properties to algorithm

**Exceptions:**

**Step 1:** Rules may be altered later in the algorithm process due to dysfunctionality

**Post conditions:** Settings are applied to the algorithm

**Priority:** 2\*

**ID:** RG2

**Name:** Tile Placement

**Summary:** This use case it responsible for placing the tiles based on the array given by the algorithm

**Preconditions:** Must be given the array with the play area from the algorithm

**Basic sequence:**

**Step 1:** Access array contents including characters

**Step 2:** Choose tile set from Tileset case use

**Step 3:** Scale the 1x1 ascii units to custom scale tile preset

**Exceptions:** None

**Post conditions:** Floor tiles have now created spawn-able platforms for the player and enemies

**Priority:** 1\*

**ID:** RG3

**Name:** Prefab Placement Bounds

**Summary:** These bounds are given by tile place and allow for objects such as the player, enemies, furniture

**Preconditions:** There must be floor tiles in the play area. This doesn’t have to be made by the Algorithm and Tile Placement use cases since it’s universal, meaning hand made levels may also be used by this

**Basic sequence:**

**Step 1:** Access all floor tiles’ x and y positions in the play area

**Step 2:** Give possible positions to all spawning objects

**Exceptions:**

**Step 1:** If no prefabs need to exist in the play area, or in certain rooms, this will not be necessary

**Post conditions:** Applies to external objects

**Priority:** 3\*

**ID:** RG4

**Name:** Tileset

**Summary:** The chosen set of tiles to be used for the Tile Placement use case

**Preconditions:** Must have a Tile Placement algorithm to be placed

**Basic sequence:**

**Step 1:** Access all tiles sets

**Step 2:** Allow the editor to choose from all the tiles

**Exceptions:**

**Step 1:** Not all tile sets must be used

**Post conditions:** None

**Priority:** 2\*

**ID:** RG5

**Name:** Weapon Inventory

**Summary:** Keeps track of the two weapons the player currently has.

**Actors:** Player

**Preconditions:** Must have two gun slots in order to receive guns

**Basic sequence:**

**Step 1:** Check if player has weapons (allows for gun cycling)

**Step 2:** Allow guns to be rearranged, switched, or removed.

**Step 3:** Allow for only two guns maximum.

**Exceptions:**

**Step 1:** Do not allow guns cycling if player head less than 2 guns.

**Post conditions:** None. Player does not have to have a weapon for this to work properly.

**Priority:** 2\*

**ID:** SM1

**Name:** Player Funds

**Summary:** This keeps track of the players funds/coins.

**Actors:** Player

**Preconditions:** None

**Basic sequence:**

**Step 1:** Player picks up currency (coins) which are added to a global variable.

**Step 2:** Allow the player to spend currency at vending machines.

**Exceptions:**

**Step 1:** Player must have sufficient funds in order to buy items.

**Post conditions:** None

**Priority:** 2\*

**ID:** SM2

**Name:** Active Abilities

**Summary:** Gives the player passive power ups that stack

**Actors:** Player

**Preconditions:** Must acquire power up objects across the level

**Basic sequence:**

**Step 1:** Player picks up a power up.

**Step 2:** The special ability is applied to the player.

**Step 3:** These abilities may stack on top of each other.

**Exceptions:** None

**Post conditions:** Player becomes more and more powerful.

**Priority:** 3\*

**ID:** SM3

**Name:** Shop UI

**Summary:** This system is responsible for showing the player the trade deals for weapons and other buyable products in the shop.

**Actors:** User Interface

**Preconditions:** Player must activate a vending machine for this to occur

**Basic sequence:**

**Step 1:** Show the player all the items for sale at that specific vending machine.

**Step 2:** Allow the player to purchase an item.

**Step 3:** Allow the player to select a previously purchased gun.

**Exceptions:** None

**Post conditions:** None

**Priority:** 2\*

**ID:** SM4

**Name:** Purchase

**Summary:** This is the processes of removing currency from the player in exchange for an item

**Actors:** Weapons, Special Abilities, etc.

**Preconditions:** Must be able to make the purchase in order to receive the item

**Basic sequence:**

**Step 1:** Let the player choose what they want to purchase.

**Step 2:** Make sure they have enough funds for said item.

**Step 3:** Give them the item or allow them to equip it.

**Exceptions:**

**Step 1:** Do not make the purchase if the player has insufficient funds.

**Post conditions:** The player is given a new item.

**Priority:** 1\*

**ID:** SM5

**Name:** Active Abilities

**Summary:** This system keeps track of all the abilities the player has and how much of them they have.

**Actors:** Player

**Preconditions:** None. May be empty.

**Basic sequence:**

**Step 1:** Check for players active abilities.

**Step 2:** Apply them to the player.

**Exceptions:**

**Step 1:** If the player has none, apply none.

**Post conditions:** The player has abilities to help them in the game.

**Priority:** 2\*

**ID:** SA1

**Name:** Add Abilities

**Summary:** This adds an ability to the player

**Actors:** Special Abilities (List)

**Preconditions:** Must be picked up or bought

**Basic sequence:**

**Step 1:** Player retrieves power up.

**Step 2:** Adds actual ability to player.

**Exceptions:**

**Step 1:** None. These may stack.

**Post conditions:** The Player is given an ability.

**Priority:** 2\*

**ID:** SA2

**Name:** Stack Abilities

**Summary:** This stacks an ability the player already has

**Actors:** Special Abilities (List)

**Preconditions:** Must already have the ability in question

**Basic sequence:**

**Step 1:** Player picks up/buys ability.

**Step 2:** Check to make sure the player already has the ability.

**Step 3:** Stack the ability onto the existing stack, increasing its effectiveness.

**Exceptions:**

**Step 1:** Do not stack if the player does not have the ability already.

**Post conditions:** The effectiveness of the said ability is increased.

**Priority:** 2\*

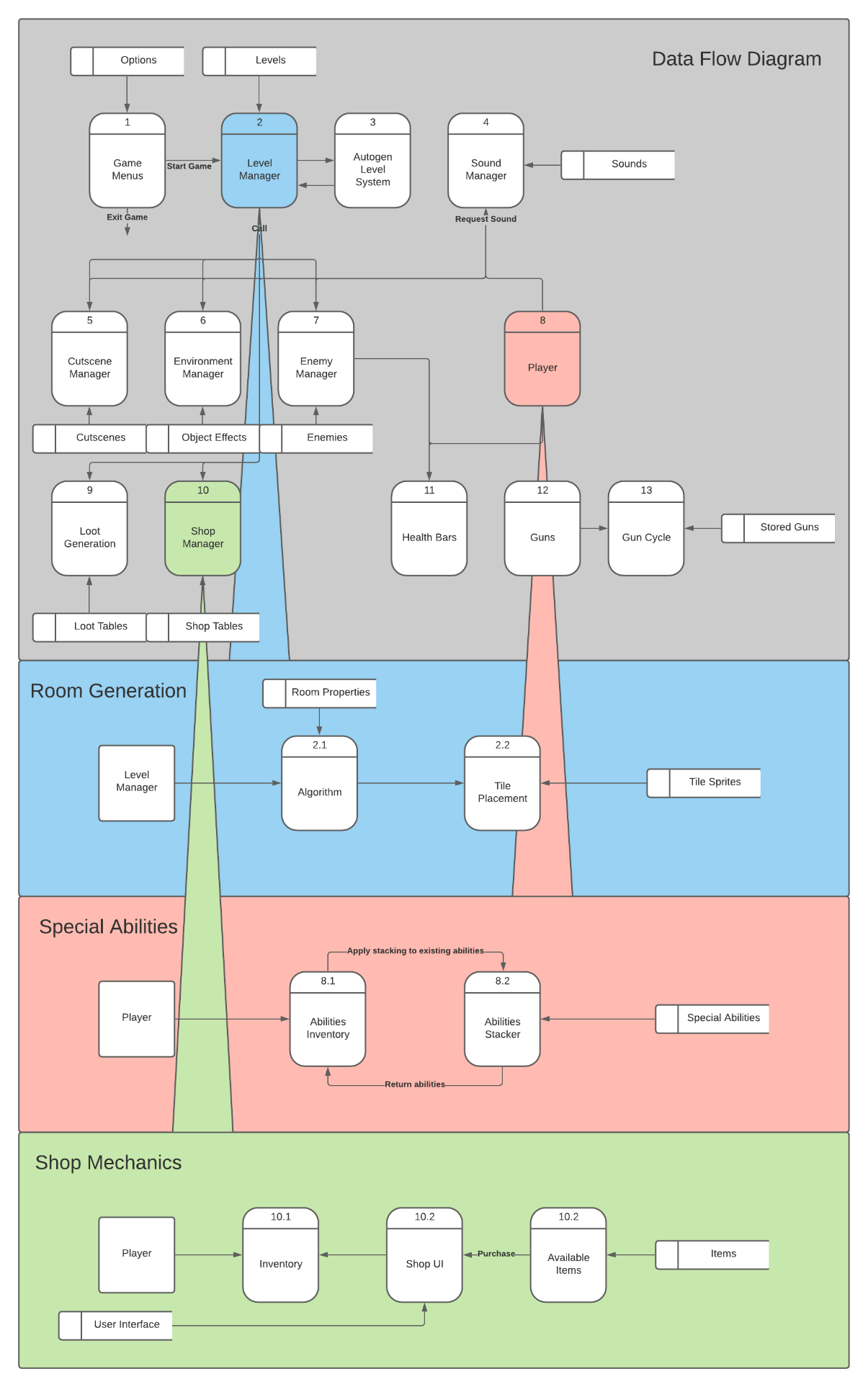
**ID:** SA3

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

[Get the Level 0 from your team. Highlight the path to your feature]

Example:

### Data Flow Diagrams



### Process Descriptions

**Room Generation**

Algorithm:

WHILE the output of the room generation is false

Continue to generate until a legal play area is defined with Room Properties

END WHILE

Tile Placement:

IF the Algorithm has determined a successful play area,

Place tiles given from Tile Sprites

END IF

**Special Abilities**

Abilities Inventory:

IF the player has abilities

Show them in inventory/Allow them to be used

END IF

Abilities Stacker:

IF the player has abilities and the same abilities are added to them

Add them/Stack them onto the player’s abilities

END IF

**Shop Mechanics**

Inventory:

IF the player has items in their inventory

Display them or apply them to the player

END IF

Shop UI:

IF the player interacts with a shop

Display all the available items and allow them to purchase them

END IF

Available Items:

IF there are items given to this class

Send them to the SHOP UI for the player to purchase

END IF

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

**Room Generation Tests**

The Room Properties of the Algorithm are the inputs I will be changing and testing in order to stress test my Algorithm. The output, put simply, should return a playable room. These tests will also make sure Tile Placement also works correctly, and that there are no exceptions to the rules I apply.

Room Properties:

* Total Play Area Width
* Total Play Area Width
* Door amount (Hallway amount)
* Room Amount
* Min Room Size
* Max Room Size

**Shop Mechanics Tests**

This set of testing should be fairly simpler than stress testing the Room Generation Algorithm. This is why I’ve included the tests with the actual development of the feature in the timeline section. With these tests, I will simply make sure the Shop UI displays all of the items we require and that the player is able to purchase items if they have enough currency. Of course, if the player does not have enough currency, it will not let them purchase the item.

**Special Abilities**

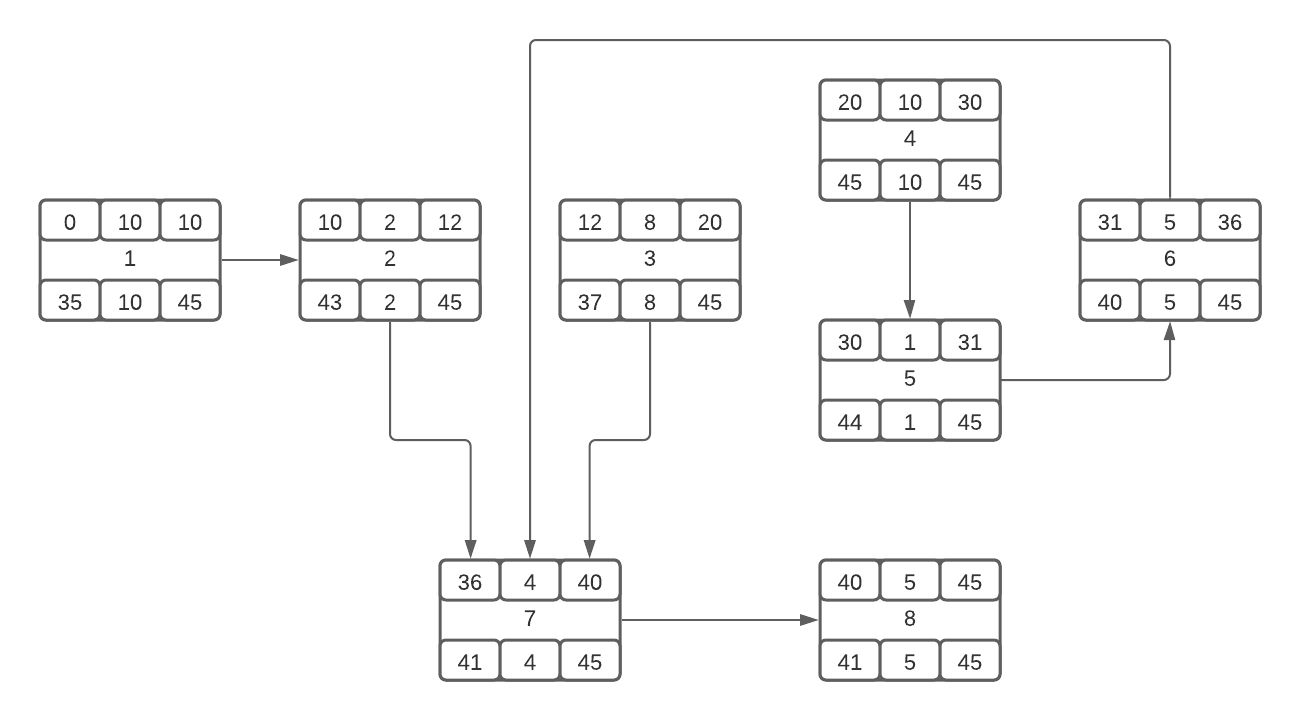
A large bulk of the testing will mostly be with creating and testing the actual special abilities for the player. The easiest part of this feature will definitely be implementing player pickups and multipliers for each ability.

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

### Work items

|  |  |  |
| --- | --- | --- |
| Task | Duration (Hours) | Predecessor Task(s) |
| 1. Room Generation | 9 | - |
| 2. Room Generation Stress Tests | 3 | 1 |
| 3. Shop Mechanics (Tests also) | 8 | - |
| 4. Special Abilities (Abilities) | 10 | - |
| 5. Special Abilities (Stacking) | 1 | 4 |
| 6. Special Abilities (Testing) | 5 | 5 |
| 7. Code Upload/Download | 4 | 2, 3, 6 |
| 8. Combining Code (w/ Team) | 5 | 7 |

### Pert diagram



### Gantt timeline

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 |  |