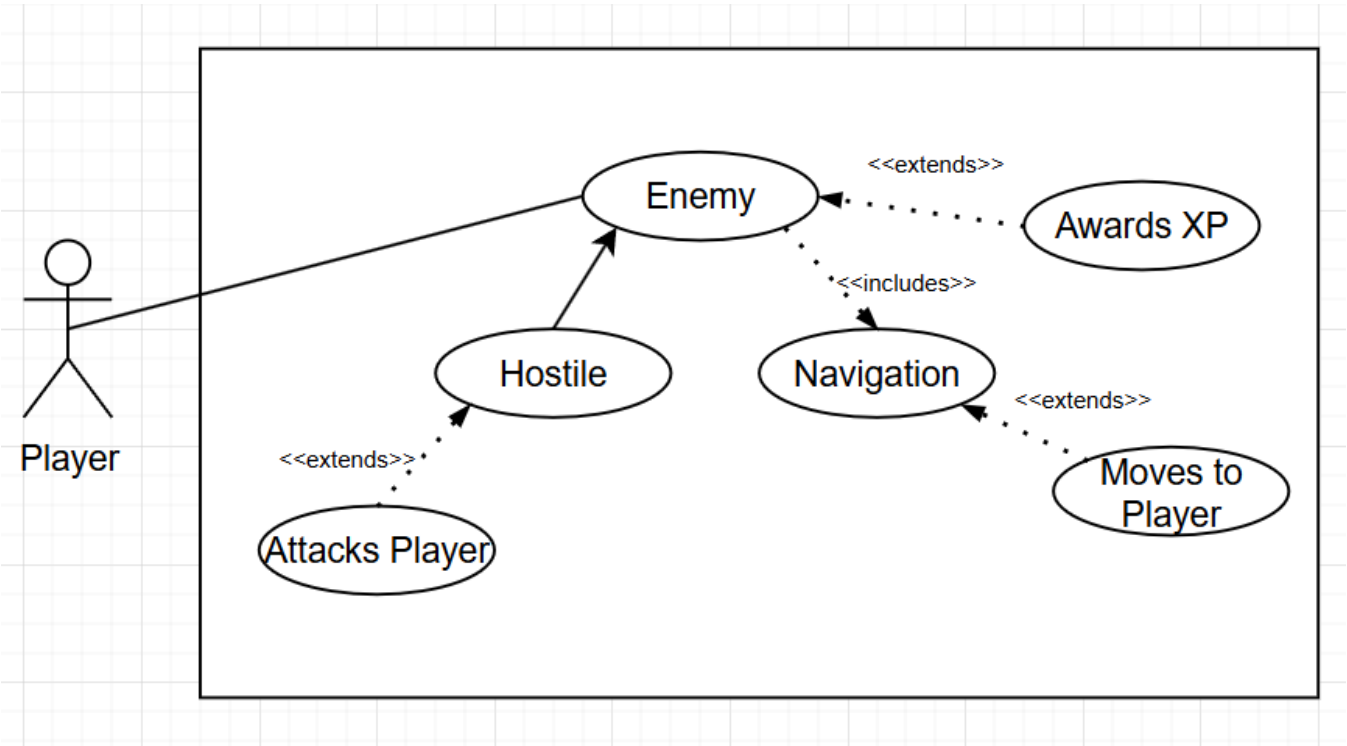
Name Gabriel Roy Mark __/50

## 1. Brief introduction __/3

For my feature, I am responsible for all things enemies inside the game (except for bosses). I will be creating all the enemy types, their attacks, movements, sprites, and sounds. Enemies will be the main factor in determining if this game will be engaging, so it is critical I ensure that they are properly executed and developed.

## 2. Use case diagram with scenario __/14




## Scenario

**Name:** Enemy Encounter.
**Summary:** Accounts for enemy interactions with the player.
**Actors:** Player.
**Preconditions:** Player is inside a room with enemies.
**Basic Sequence:**

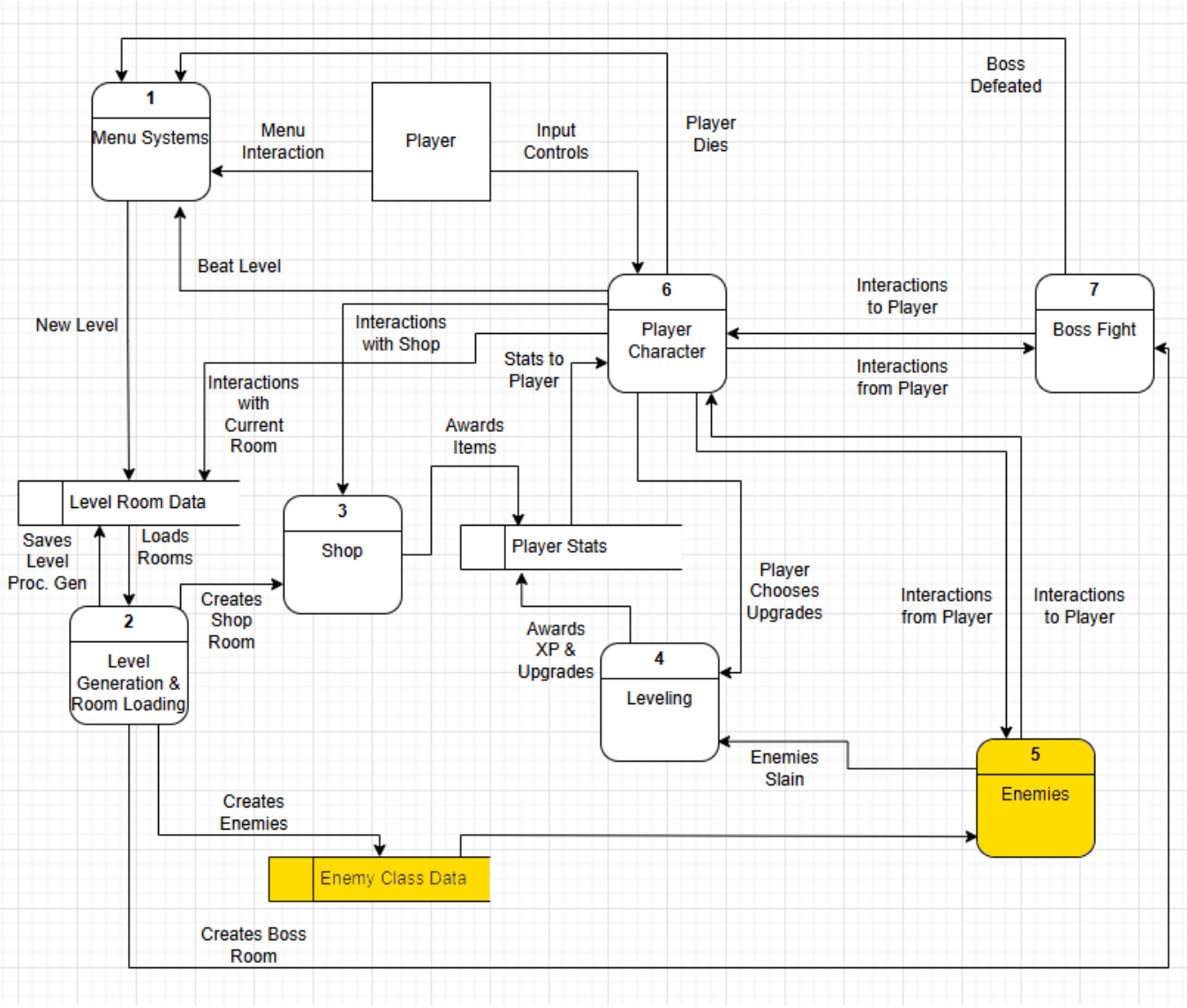
- **Step 1:** Enemy wanders the room.
- **Step 2:** Enemy detects player.
- **Step 3:** Enemy moves too player.
- **Step 4:** Enemy attacks player.

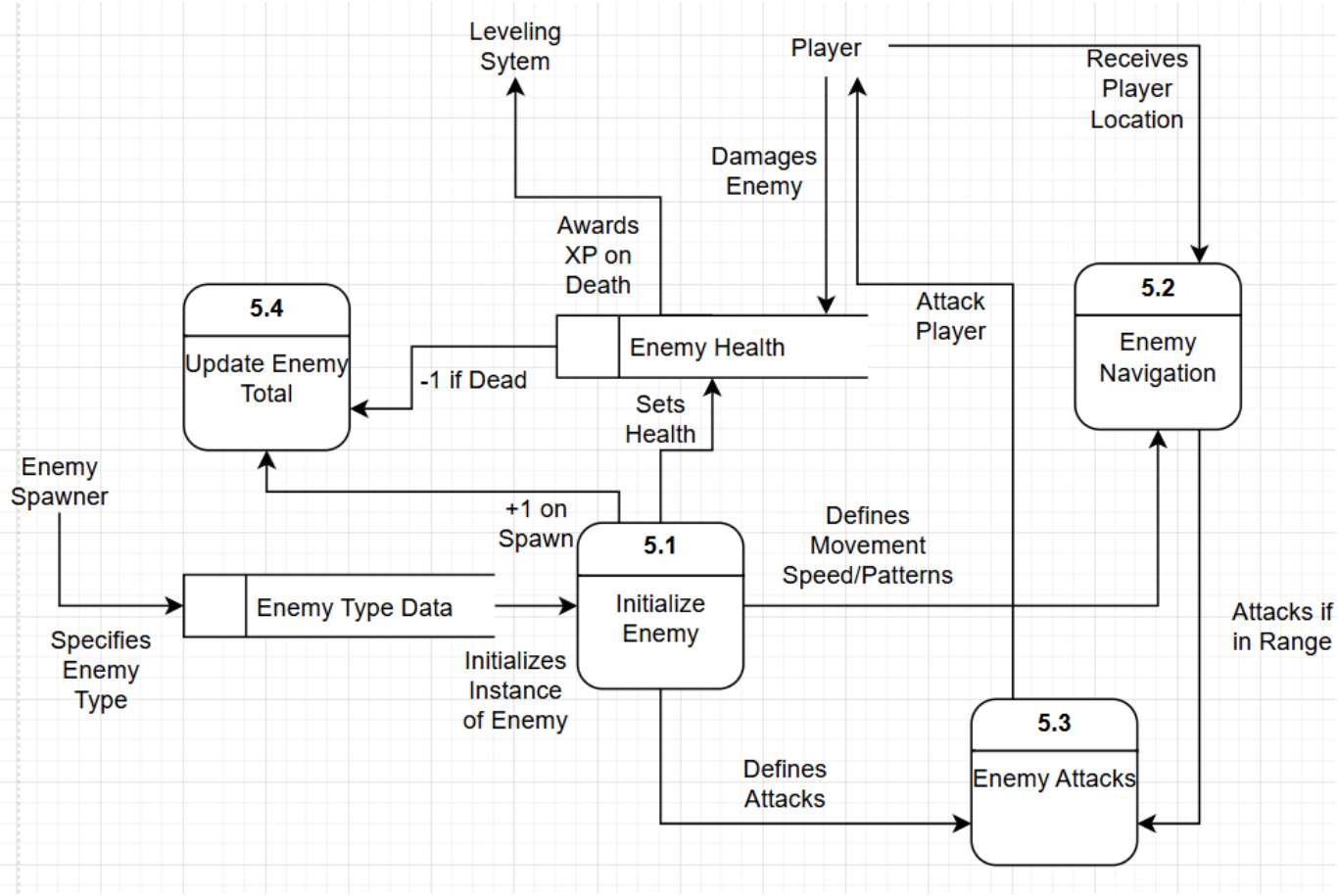
**Step 5:** Player kills enemy.
**Step 6:** Enemy awards player XP.

**Exceptions:**
**Step 1:** Enemy never detects player.
**Step 2:** Player dodges enemy attacks.
**Step 2:** Enemy kills player.

**Post Conditions:** Enemy encounter is over.
**Priority:** 1*
**ID:** EM1
*The priorities are 1 = must have, 2 = essential, 3 = nice to have.

## 3. Data Flow diagram(s) from Level 0 to process description for your feature __/14






## Process Descriptions

**Initialize Enemy:** The subclass contains all the information that makes the specific type of enemy (skeleton, spider, etc…) unique. The level designer will put down points for enemies to spawn when a room is first entered into and when the enemy spawns in is when this process happens.

*Zombie example:*

```
int health = 100;
int movementSpeed = 5;
int xpAward = 5;
string name = "Zombie";
```

*Vampire Example:*

```
int health = 200;
int movementSpeed = 10;
int xpAward = 20;
string name = "Vampire";
```

**Enemy Navigation:** Uses the player's location to determine the enemy movement, if the player is far away the enemy will not notice them. When in range or line of sight of the player, the enemy will approach the player so it can start attacking them.

```
if PlayerDistance < 10 or FacingPlayer == true then
  Move to player
else
  Continue roaming
```

**Enemy Attacks:** Uses appropriate attacks if in range of the player, for example, skeletons will only perform melee attacks when in close proximity with the player.

```
if PlayerDistance < 1 then
  Perform a melee attack
Else if PlayerDistance < 6 then
  Perform a ranged attack
```

## 4. Acceptance Tests __/9

There will be several tests for the enemies to ensure proper optimization and error handling. Each test will be run individually for each different enemy type and will run for 2 minutes.

- Spawn 100 enemies in the same room, along with an invincible player.
- Spawn 100 enemies in the same room, along with a non-invincible player.
- Spawn 100 enemies in the same room, no player.
- Spawn 100 enemies out of bounds, invincible player inside room.
- Spawn 100 enemies out of bounds, non-invincible player inside room.
- Spawn 100 enemies out of bounds, no player.

For example, this would be the output table for the first bullet point when all the enemy types have completed the first test:

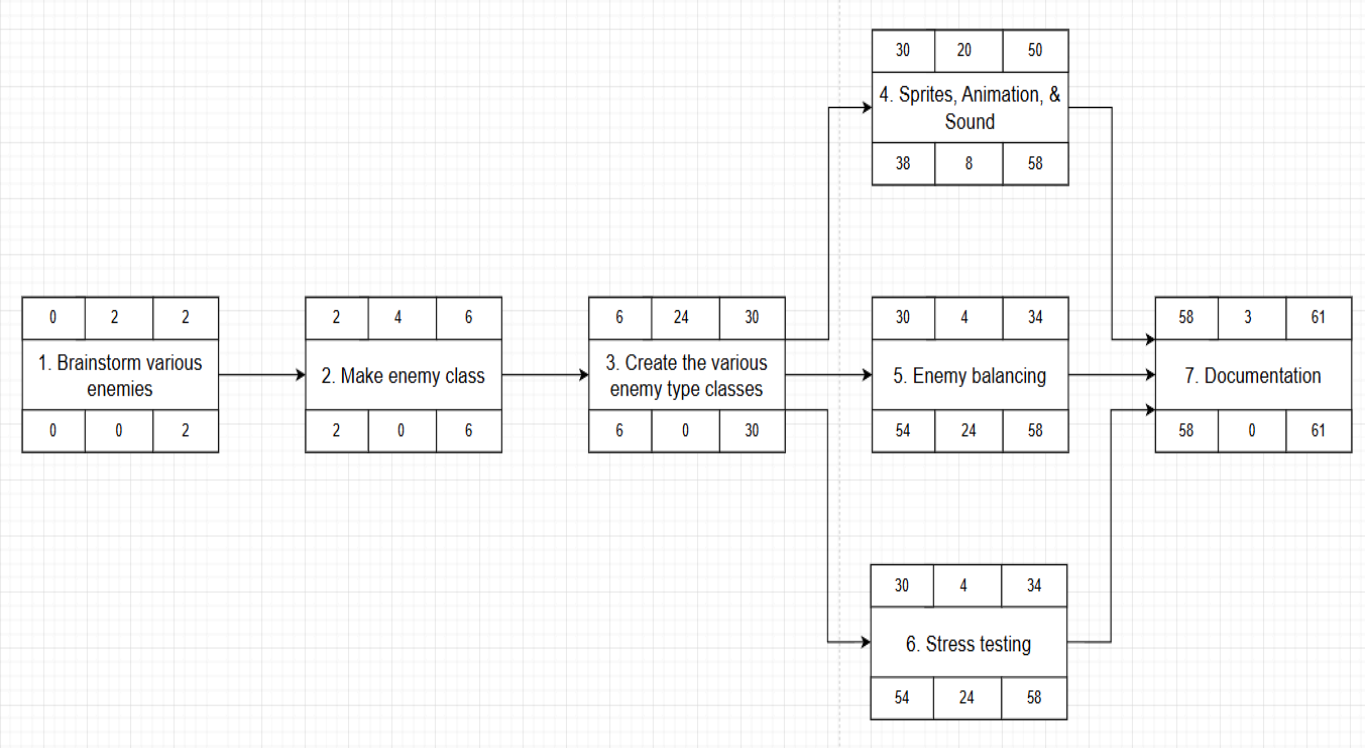
| **Enemy** | **Game Crashed?** | **Average FPS** | **Bugs Arise? (Describe if so)** |
|---|---|---|---|
| Enemy Type 1 | No | 37 | No |
| Enemy Type N-1 | Yes | 4 (before crashing) | No |
| Enemy Type N | No | 48 | Enemies were able to kill each other. |

## 5. Timeline __/10

### Work Items

| Task | Duration (hours) | Predecessor Task(s) |
|---|---|---|
| 1. Brainstorm various enemies | 2 | - |
| 2. Make enemy class | 4 | 1 |
| 3. Create the various enemy type classes | 24 | 2 |
| 4. Sprites, Animation, & Sound | 20 | 3 |
| 5. Enemy balancing | 4 | 3 |
| 6. Stress testing | 4 | 3 |
| 7. Documentation | 3 | 4,5,6 |

### Pert diagram



# Gantt timeline