Name Sam K Mark _____/50

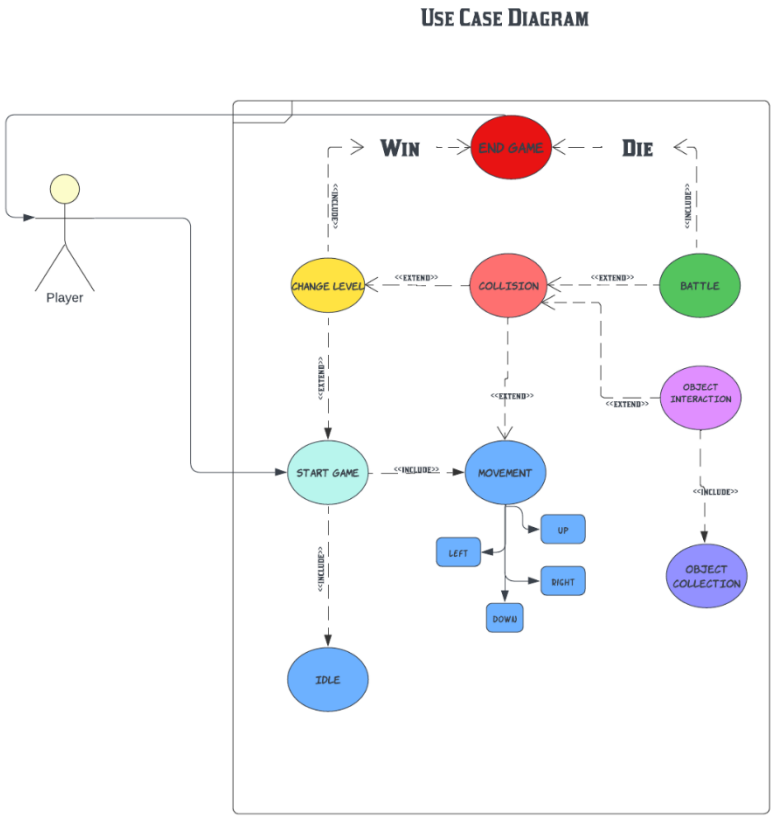
1. Brief introduction

Feature: MVP and Action Event Manager

Our game is an escape room where the player must get out of all the rooms to win. As a minimum viable product developer, I will be creating player movements, camera movement and collisions. The purpose of making the minimum viable product is to create a base from which everyone else can start adding their own features.

I will also be handling basic level management. This could be included in the MVP, but level management is “technically” optional. The game will work and be fun without having various levels. The levels are different rooms that the player needs to get out of. There will be 3 levels, or 3 rooms; each outside the other; once the player gets out of all 3 rooms, they will be victorious.

2. Use Case and scenarios



Scenarios

Name: Start Level
Summary: The level starts, it could be the first level or the next depending on whether the player just started the game or if the player just beat another level.
Actors: Player
Preconditions: New game or if the player just beat another level.
Basic sequence:

Step 1: If the unit is not moving then they will stay idle.
Step 2: If W, A, S, D or ***up Arrow, down Arrow, left Arrow, right Arrow*** is pressed then the unit moves in the direction of that input.

Exceptions:

Step 2.1: A button other than the above input is pressed: stay Idle.
Step 2.2: If the unit collides to another object, then the unit will stop moving.

Post conditions: The unit moves or stays idle.
Priority: 1*
ID: C01

Name: Object Interaction
Summary: When the unit collides to an object, they can interact with the object.
Actors: Player
Preconditions: The player must collide to an object.
Basic sequence:

Step 1: If the unit collides with an object, they can interact with it.
Step 2: When interacting with an object, the unit can collect that object to progress through the game.

Exceptions:

Step 2: The unit can choose not to collect the object.

Post conditions: Collection of objects, progressing further in game.
Priority: 1*
ID: C01

Name: Level Change
Summary: The unit gets to the next level (*room*) if they can complete the current level.
Actors: Player
Preconditions: The player must collide with the door after completing the level.
Basic sequence:

Step 1: The door can be unlocked if the player completes the level (*finds all the clues, or other methods not yet finalized*)

Step 2: The player can get to the next level after interacting with the door they collided with.

Exceptions:

Step 2: The game ends if the current level is the final level.

Post conditions: Get to the next level or win game.
Priority: 1*
ID: C01

Name: Start Battle
Summary: When the unit collides with the **range of view** of an enemy, the unit can battle with the enemy using various methods.
Actors: Player
Preconditions: The player must be near an enemy.
Basic sequence:

Step 1: If the unit is within the enemy's field of view, the unit can battle the enemy.

Step 2: The battle is won if the player successfully defeats the enemy using various methods.

Step 3: If the player fails to defeat the enemy, they **die.**

Exceptions: None (*The player cannot run if the battle commences*)
Post conditions: The player gains hints or the game **ends**.
Priority: 3
ID: C03

Name: End Game
Summary: The game ends if certain criteria is met.
Actors: Player
Preconditions: Player must complete the **final** level, or the player must **die**.
Basic sequence:

Step 1: The game ends if the player interacts with the final door, or if the player dies.

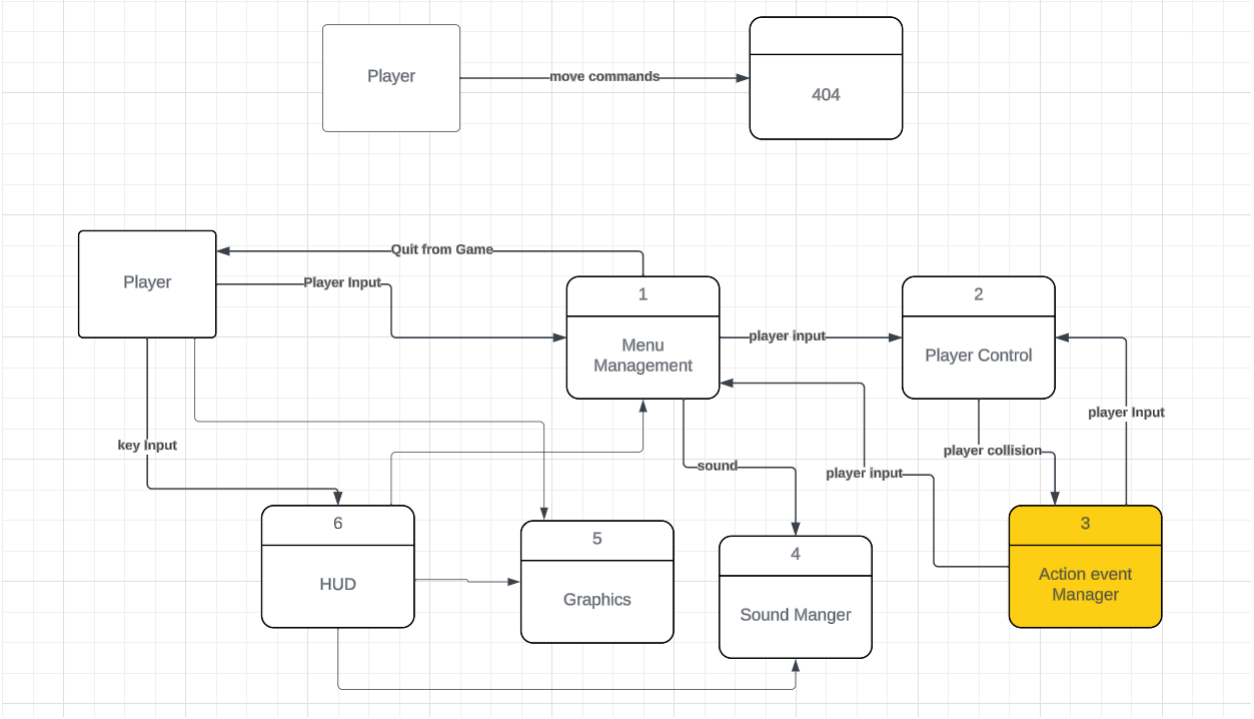
Step 2: If the game ends with the final door interaction: **the game is won**. If the game ends with the player's death: **the game is lost**.

Exceptions: None

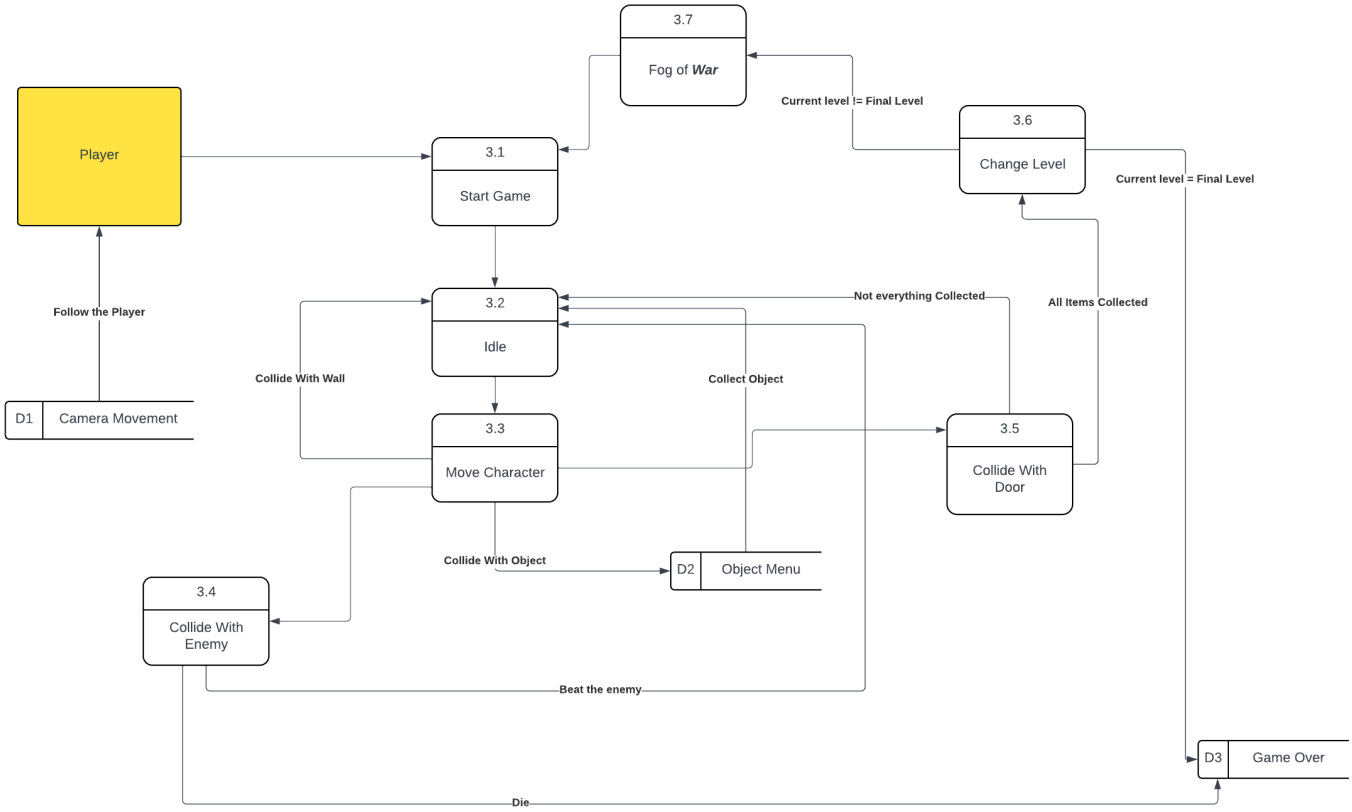
Post conditions: The game is either **won** or **lost**.
Priority: 1*
ID: C01

3. Data Flow diagrams

Level 0 Diagram



Level 1 Diagram (Action Event Manager)



Process Description

Start Game (3.1) process will start the game:

If(**Start Game**):

Initialize Player Position:

Position - (0, 0)

idle (3.2)

Initialize **Camera Movement (D_1)**

Initialize Object Positions;

Idle (3.2) process occurs after the game has started. This will ensure that the player is Idle and not moving when the game starts.

Move Character (3.3) process occurs when Movement input has been triggered. This process has a few post conditions. When the player collides with a wall, they simply stay Idle whereas when the player collides with an enemy, door or other objects, a few conditions are triggered.

Collide with Enemy (3.4) process occurs when the Player collides with an enemy. The player will only have the choice to fight when this happens. If the player beats the enemy then they **may** get some rewards (*not yet decided on this feature)* and then they stay idle, however if the player loses then they **die** and the game ends.

if(Collision == Wall):

Idle (3.2)

if(Collision == Enemy):

If(Battle == Won):

idle (3.2)

else:

Game Over ($D_{3)}$;

if(Collision == Door):

Collide with Door (3.5);

Collide with Door (3.5) process occurs when the Player collides with the room's door. If the player collides with the door after collecting all the items and/or solving all the puzzles then the door opens and the level changes, however, if the player collides with the door while all of the tasks aren't complete then they just stay idle.

if(**Collide with Door**):

if(Level Complete):

Change Level (3.6)

else:

idle (3.2)

Change Level (3.6) process occurs when the Player collides with the door after collecting all the items and/or solving all the puzzles. There are 2 post conditions for this process, if the current level that the Player just completed is not the **final level** (*most likely level 3)*, then the Player is sent to the **Start Game** of the next level, however, if the Player just completed the final level then the Game Ends.

if(**Change Level**):

if(next level != NULL):

current level = next level;

else:

Game Over (D_{3});

Fog of War (3.7) process occurs when the Player leaves the current Room (*level)* and gets to the next level. In video games, Fog of War is referred to the parts of a map that you can't see due to it being in a lack of your field of view. For example, in most top-down 2D games, you will not be able to see on top of mountains due to it being higher than your field of view. **For our game, Fog of War occurs in a room that the player is currently not in.** Since our game contains layers of rooms outside one another, the player will not be able to see inside a room that they aren't currently inside. Suppose the player is in

Room 2; they will not be able to see inside Rooms 1 or 3. Although **Fog of War** is a term mostly used in **War Strategy** games, I've always loved this so I figured I'd use it; I guess we could also call this the **See-able Area.**

if(current Room == n):

    Fog(n) = Disable;

Camera Movement ($D_{1)}$ contains the information on the Player's position so it can follow the Player.

Object Menu ($D_{2)}$ contains the option to collect an object.

Game Over ($D_{3)}$ contains the End Game scene.

4. Acceptance Tests

Bugs – Errors that would cause players to not be able to play the game how it was meant to be played.

Void – A deep pit where players fall into due to glitch in the matrix. (*or in boring words, outside the unity's camera lens)*

FPS – Frames Per Second, or a gamer's worst nightmare.

Since I play a lot of games myself, I feel like I know what to check for in our video game. I will be testing out the movements and level changes. I will make sure that there are no glitches like getting stuck in walls, wrong level change (for example, you might glitch into level 3 after level 1, instead of level 2). I will also be making sure that object interactions aren't bugged by interacting with the objects through every possible way. I will also help check whether there are collision problems with the enemy, since the enemy has a certain field of view, it might glitch and keep walking back and forth due to certain positioning exploits. My overall job is to ensure that the gameplay experience does not have loopholes that players can exploit.

The starting point for the Acceptance test would be a stress test, I will keep increasing the player's movement speed and set a starting position and a unidirectional movement to see if the player would pass through the wall.

The second Acceptance test is to collide with the door to change the level multiple times to see whether the player would glitch into the void. This could happen if the level manager cannot close before another instance of it opens. This would also not happen in-game because a player cannot change levels that quickly, but it is also still good practice.

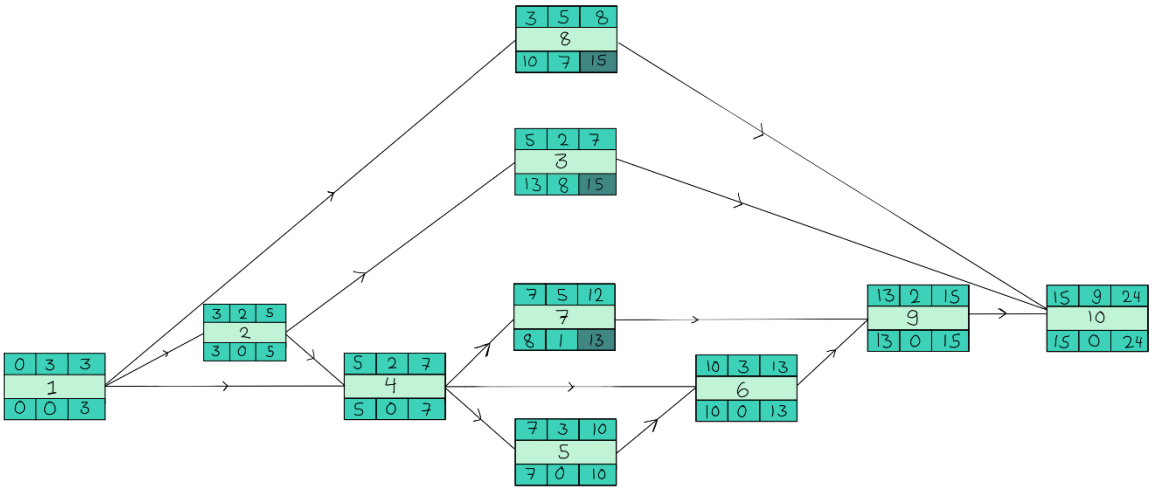
The last test would be to create about 100 objects and try interacting with all of them to see if the game crashes. This would be helpful to check how many objects we could add before the FPS starts dropping below a certain limit.

5. Timeline

Task	Duration (Hours)	Predecessor Task(s)
1. Room Creation	3	-
2. Player Construction	2	1
3. Camera Control	2	2
4. Collision Detection	2	1, 2
5. Object Interaction	3	4
6. Level Management	3	4, 5
7. AI Logic	5	4
8. Extra Functionalities	5	1
9. Improve Gameplay/ FPS	2	6, 7

(Frames Per Second)		
10. Testing/Debugging	9	3, 8, 9

Pert diagram



Gantt timeline