Name: Jillian Jabbes Mark _____/50

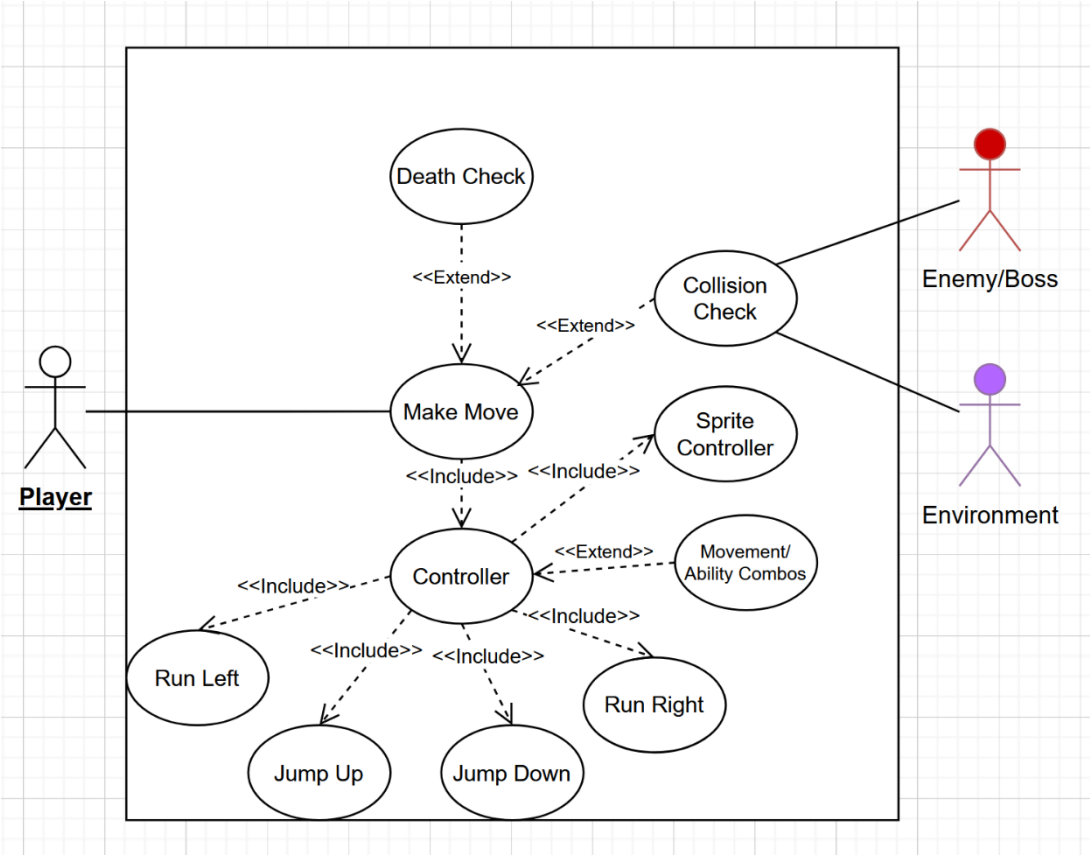
1. Brief introduction __/3

For BrainStew's 2D game project, I am the developer responsible for the Player System. This role involves designing and implementing core playable character mechanics including movement, combat, and physics interactions with the environment (object physics). Movement specifically would mean defining the default controls such as idle, running, and jumping, along with other abilities like dashing and wall-running. Then playtesting will help determine if other mechanics may be useful like sliding. I will also be working closely with our weapon and powerups specialist to make sure those integrate well with the player's combat. Additionally, sprite rendering and animation would also fall under this category so all actions are responsive to player input (and look cool).

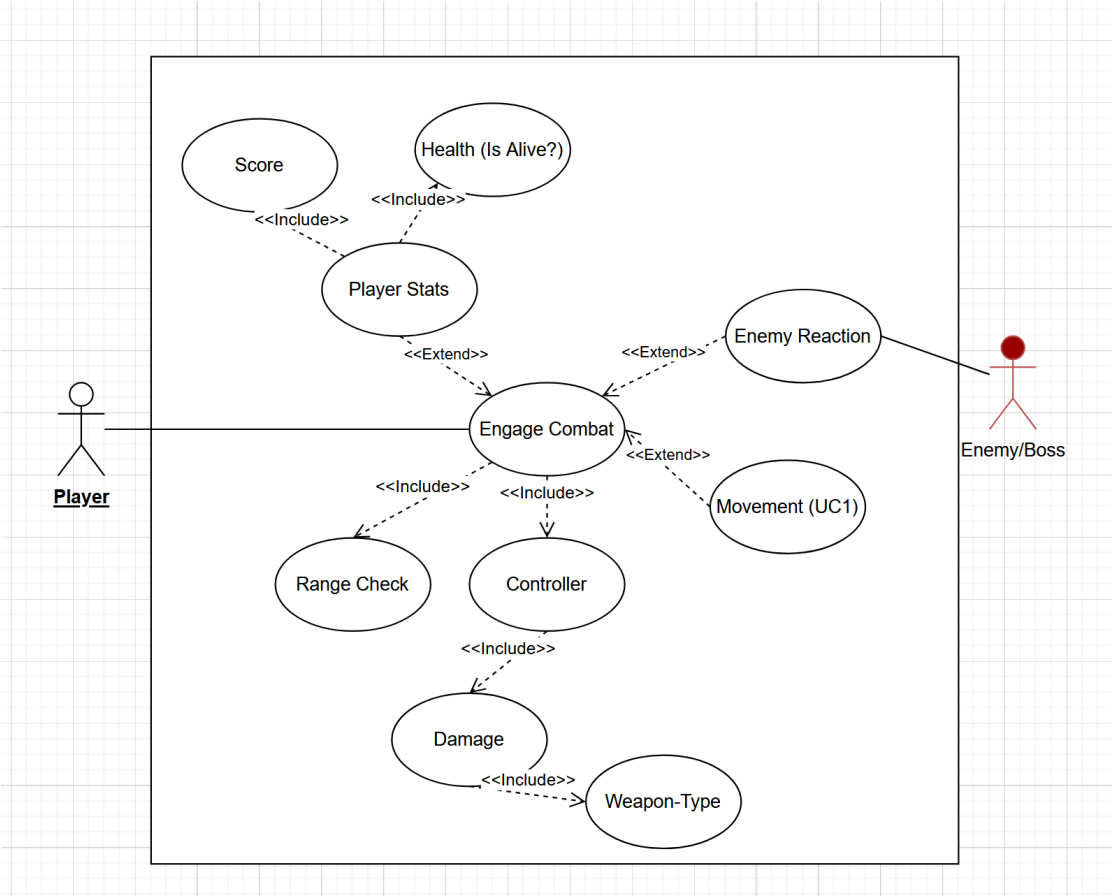
Beyond mechanics, working on the player system would mean I must define the player stats encompassing health, inventory, and abilities (or if there are temporary effects). It would feed real-time data to the UI/HUD so that whoever is playing the game is able to see what their health is currently at or inventory contents.

2. Use case diagram with scenario __14

Use Case 1: Basic Movement



Use Case 2: Player Combat



Scenarios

Name: Use Case 1: Basic Movement
Summary: Player wants to move the character. They send input to navigate a 2D environment while having conditions of being impacted by collisions or when the character dies
Actors: Player (Primary), Enemy/Boss & Environment (Secondary)
Preconditions: Level scene and relevant game objects including the player object and manager itself. Player health > 0. Movement controls enabled (not in menu mode)
Basic sequence:

Step 1: Continuous health check
Step 2: User/Player presses defined trigger keys for movement

Step 3: System validates input against control scheme. If correct, movement is called and input is mapped to defined action
Step 4: Action is called

Step 5: Character object moves according to input factoring in defined input combos or basics

Step 6: Movement executed by controller also includes signals sprite animation and to update accurately to movements made

Step 7: Always during movement the player object's physics checks for collision with other objects like environment or bosses and enemies

Exceptions:

Step 1: Dead state (health <= 0) means disable inputs

Step 2: Invalid input

Step 3: Collision with enemy/boss or environment

Post conditions: Character object moves accurately in response to player input, position updated in world space, animation state matches movement intent, collision responses applied, and when inputs cannot occur (no hp)

Priority: 1

ID: UC1

*The priorities are 1 = must have, 2 = essential, 3 = nice to have.

Name: Use Case 2: Player Combat

Summary: Player engages with hostile enemies using a mode of attack by sending the combat trigger/input to initiate. They can use their current weapon to damage enemies

Actors: Player (Primary), Enemy/Boss (Secondary via hitbox)

Preconditions: Level scene and relevant game objects including the player object and manager itself. Enemy within defined range. Player health > 0.

Basic sequence:

Step 1: User/Player presses trigger key for attack (character uses weapon to strike and peripheral to aim)

Step 3: Continuous health check

Step 2: System validates input against control scheme. If correct, attack is called and input is mapped to defined action

Step 3: Action is called

Step 4: Attack executed by controller also includes signals to sprite animation and to update accurately to movements made

Step 4: If weapon hits enemy hitbox, enemy has reaction namely losing hp.

Step 5: Damage done depends on what weapon being used and enemy reactions to such attack

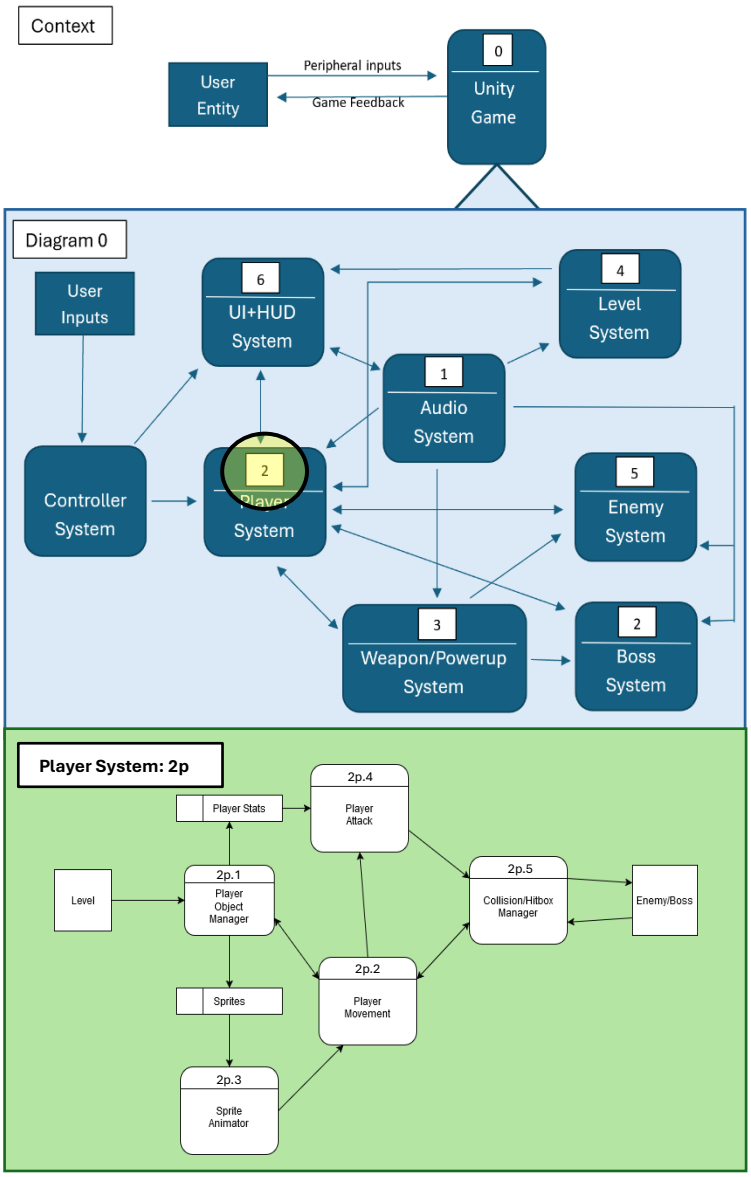
Exceptions:

Step 1: Health is <= 0 so cannot engage in attack. Must be redirected via UI
Step 2: Invalid input
Step 3: Enemy reaction (their attack) interrupts player attack
Step 4: Not in range
Step 5: Knockback

Post conditions: Character engages and attacks in combat accurately to player input and defined exceptions in game world. Player or enemy or both loses health.
Priority: 1
ID: UC2
*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

TechDown: Data Flow Diagram Abstraction Levels



Process Descriptions

(2p.1) Player Object Manager

Manages core player states, communicates with stats, movement, and animation...

```
function CreatePlayer():
        LoadPlayerIntoWorld()
        LoadPlayerStats()
        SetupDependencies()
end
function LoadPlayerIntoWorld():
        SpawnPlayerAtSpawnPoint()
        InitializeSprite()
        LoadPlayerStats():
end
function SetupDependencies():
        LinkMovementSystem()
        LinkCombatSystem()
        LinkAnimationSystem()
End
```

(2p.2) Player Movement

Would handles movement input, physics, and collision detection...

```
function MovePlayer():
  velocity = GetInput()
  ApplyGravity()
  if CheckCollision():
        AdjustPosition()
  UpdateAnimator()
end
```

(2p.3) Sprite Animator

Syncs the animation states with movement and combat actions

```
function UpdateAnimation():
  if isMoving:
    Play("Run") //but with all movement types in 2D world
  else if isAttacking:
```

```
    Play("Attack")
  else:
    Play("Idle")
End
```

(2p.4) Player Combat

This would handles attacks, hit detection, and stat modifications.

```
function Attack():
  if CanAttack():
    PlayAttackAnimation()
    RegisterHit() //via attack medium or mode
end
```

(2p.5) Collision/Hitbox

Hit detection for movement and combat.

```
function CheckCollision():
  if DetectWall():
    StopMovement()
  if DetectEnemyHit():
    RegisterDamage()
end
```

4. Acceptance Tests _____9

Player Movement Test

Run movement 100 times each by pressing keys (desktop example) ‘w’, ‘a’, ‘s’, ‘d’

Example for movement feature

Output	Input	Notes
Jump UP	w	Character jumps up on the game world’s y-axis unless there is an obstacle/invalid input/is dead
Run LEFT	a	Character runs left on the game world’s x-axis unless there is an obstacle/invalid input/is dead
Jump DOWN	s	Character goes downward on game world’s y-axis unless there is an obstacle/invalid input/is dead

Run RIGHT	d	Character runs right on the game world's x-axis unless here is an obstacle/invalid input/is dead

Player Combat Test

After the weapon and enemy is defined, test if the character can use it to shoot projectiles with the left mouse click (desktop example) + mouseover to aim and cause damage to an enemy only when hit and if the character also takes damage only when hit. Test this 50 times both cases.

Example for combat feature

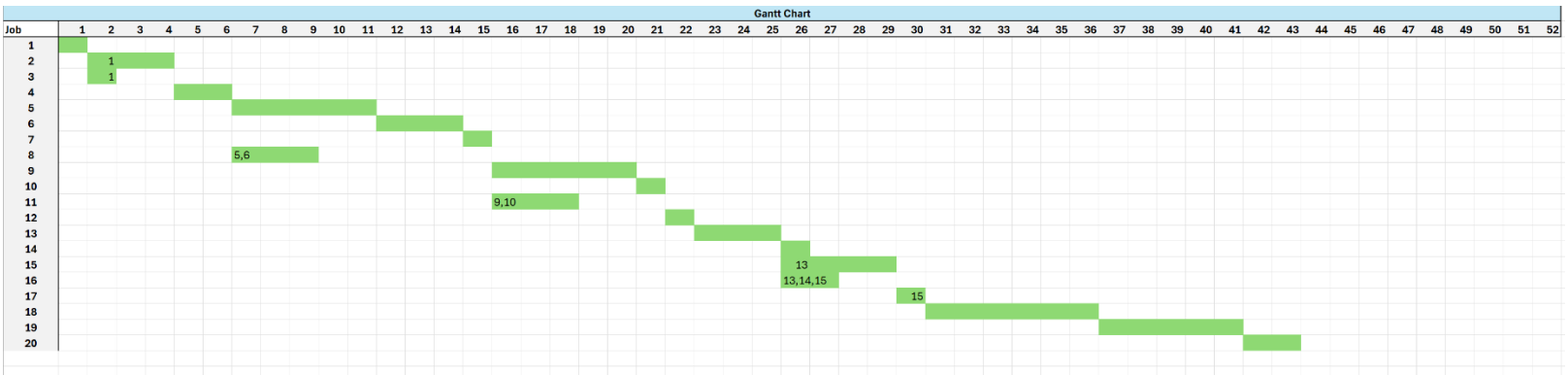
Output	Input	Notes
Shoot	Left click	Character fires a projectile from the weapon of use facing aimed direction. Follows trajectory to hit unless blocked/missed
Enemy hit	Projectile ->Hitbox	Enemy takes damage only if projectile collides with hitbox. Damage value and projectile behavior applied based on weapon stats
Player hit	Enemy attack->Hitbox	Player takes damage only if the enemy attack collides with the player's hitbox. Damage value is applied based on enemy attack stats
Missed Shot	Left Click, Out of Range	Projectile continues until max range or collides with object. No damage
Blocked Shot	Left Click, Hits Obstacle	Projectile collides with environment object and does not damage enemy
Enemy Defeated	Projectile -> HP=0	Enemy is destroyed or enters death animation when killed
Player Defeated	Enemy Attack -> HP=0	Player dies if health reaches zero then triggers death animation, respawn at last checkpoint, or game over

5. Timeline _____/10

[Figure out the tasks required to complete your feature]

Work Items		
Task	Duration (Hours)	Prerequisites
1. Define Player System Scope	1	-
2. Research Reference Games	3	1
3. Create Player Design Doc (ideas)	1	1
4. Plan Component Architecture	2	2,3
5.Setup Player Controller Script for Movement	5	4
6. Integrate Physics & Collision	3	5
7. Playtest & Adjustments	1	6
8. Setup Player Hitbox	3	5,6
9. Work with Weapon Specialist to Create Weapon Handling/Combat System	5	8
10. Playtest Shooting & Projectiles	1	9
11. Ensure Player Attacks Register	3	9,10
12. Test Combat Feel & Adjustments	1	11
13. Define Player Stats Structure	3	12
14. Implement Health System	1	13
15. Create Inventory System	4	13
16. Integrate Stats with UI/HUD	2	13,14,15
17. Test Inventory	1	15
18. Help Program Bosses	6	17
19. Implement Player Sprite Animations to Movement	5	18
20. Bug Testing & Final Playtest	2	19

Gantt Timeline



Pert Diagram