Name: **Joshua** **Norlin** Mark \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/50

## Brief introduction \_\_/3

In our pirate-themed sidescroller, dynamic and engaging enemy behavior is essential to maintaining a challenging gameplay experience. I am responsible for implementing the core logic and AI scripts that govern basic enemy movement and attack patterns.

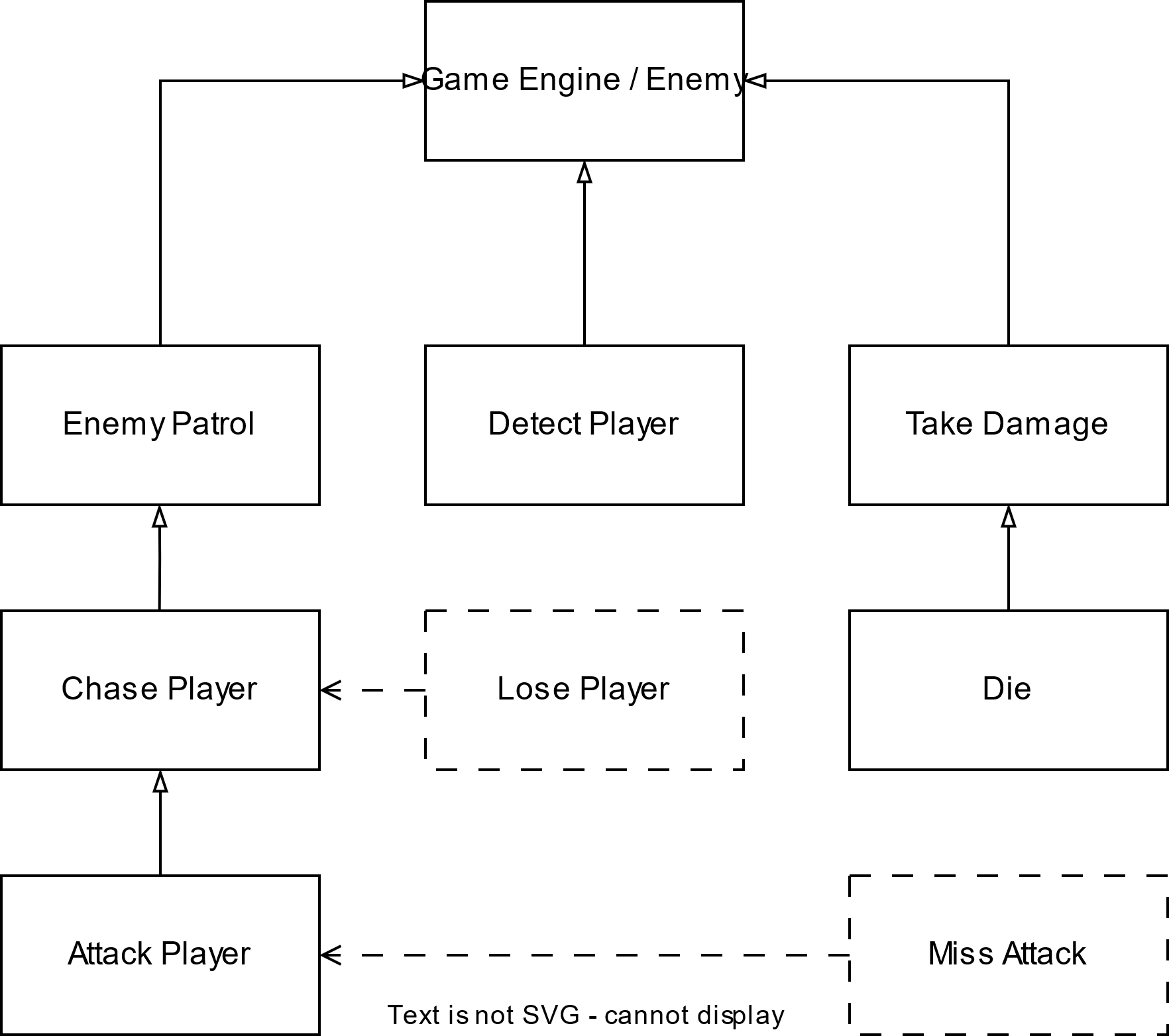
My work focuses on developing responsive, adaptable enemies that react intelligently to the player’s actions, enhancing both the difficulty curve and overall immersion. By making the code efficient and modular, the enemy behavior can be easily expanded and refined as the game evolves, providing a solid foundation for diverse enemy types throughout the levels.

I will be implementing the basic movement, attack, and miscellaneous aspects of the enemies.

## Use case diagram with scenario \_\_14

Example:

### Use Case Diagrams



### Scenarios

**Name:** Enemy Patrol

**Summary:** The enemy moves along a predefined path when the player is not detected.  
**Actors:** Game Engine / Enemy  
**Preconditions:** Enemy is initialized and the enemy is active, and no player is within detection range.  
**Basic Sequence:**

1. Enemy spawns and initializes patrol mode.
2. Enemy moves along a predefined path.
3. If the enemy reaches the end of the path, it turns around or loops.
4. Patrol continues until the player is detected.

**Exceptions:**

* **Step 2 Exception:** Obstacle suddenly appears → Enemy stops, recalculates path, or waits.

**Postconditions:** Enemy continues patrolling unless interrupted.  
**Priority:** 1 (must-have)  
**ID:** UC01

**Name:** Detect Player

**Summary:** The enemy detects the player within its field of vision or proximity range.  
**Actors:** Game Engine, Player (indirect)  
**Preconditions:** Enemy is initialized and enemy is patrolling or idle.  
**Basic Sequence:**

1. Enemy checks for player presence within detection range.
2. If the player is within range and line of sight, detection is triggered.
3. Enemy switches from patrol to chase mode.

**Exceptions:**

* **Step 2 Exception:** Player hides behind an obstacle → Detection fails.

**Postconditions:** Enemy either starts chasing the player or continues patrolling if detection fails.  
**Priority:** 1 (must-have)  
**ID:** UC02

**Name:** Chase Player **(<<extend>> from Detect Player)**

**Summary:** The enemy chases the player after successful detection.  
**Actors:** Game Engine, Player (indirect)  
**Preconditions:** Player has been detected, enemy has been intialized.  
**Basic Sequence:**

1. Enemy calculates the distance to the player.
2. Enemy moves toward the player, adjusting its path as needed.
3. If close enough, the enemy prepares to attack.

**Exceptions:**

* **Step 2 Exception:** Player moves out of detection range → **Lose Player** event triggers.

**Postconditions:** Enemy either reaches the player to attack or loses the player and stops chasing.  
**Priority:** 1 (must-have)  
**ID:** UC03

**Name:** Lose Player **(<<extend>> from Chase Player)**

**Summary:** The enemy loses track of the player, ending the chase.  
**Actors:** Game Engine, Player (indirect)  
**Preconditions:** Enemy is actively chasing the player, Players leaves detection range, enemy is initialized.  
**Basic Sequence:**

1. Player moves beyond the enemy’s detection range or breaks line of sight.
2. Enemy continues chasing for a short duration (persistence timer).
3. If the player is not re-detected, the enemy stops chasing.
4. Enemy returns to patrol mode.

**Exceptions:**

* **Step 3 Exception:** Player reappears before the timer ends → Enemy resumes chase.

**Postconditions:** Enemy resumes patrolling if the player is not found.  
**Priority:** 2 (essential)  
**ID:** UC04

**Name:** Attack Player **(<<extend>> from Chase Player)**

**Summary:** The enemy attacks the player when within range.  
**Actors:** Game Engine, Player  
**Preconditions:** Enemy is chasing the player and within attack range.  
**Basic Sequence:**

1. Enemy stops moving to initiate an attack.
2. Attack animation is played.
3. Damage is applied to the player if the attack connects.
4. Enemy resumes chasing or attacking based on player’s position.

**Exceptions:**

* **Step 3 Exception:** Player dodges → **Miss Attack** event triggers.

**Postconditions:** Attack either hits or misses, with the enemy continuing behavior based on the result.  
**Priority:** 1 (must-have)  
**ID:** UC05

**Name:** Miss Attack **(<<extend>> from Chase Player)**

**Summary:** The enemy’s attack misses the player due to dodging or movement.  
**Actors:** Game Engine, Player  
**Preconditions:** Enemy has initiated an attack.  
**Basic Sequence:**

1. Enemy performs an attack.
2. Player moves out of the attack’s hitbox.
3. Attack misses, and no damage is dealt.
4. Enemy has a brief recovery delay before it can attack again.

**Exceptions:**

* **Step 4 Exception:** Player counterattacks during the recovery delay → Enemy takes damage.

**Postconditions:** Enemy either resumes chasing or prepares for the next attack.  
**Priority:** 2 (essential)  
**ID:** UC06

**Name:** Take Damage

**Summary:** The enemy takes damage when hit by the player.  
**Actors:** Game Engine, Player  
**Preconditions:** Enemy is active, and the player performs an attack that connects.  
**Basic Sequence:**

1. Player’s attack connects with the enemy.
2. Damage calculation is performed based on attack strength and enemy defenses.
3. Enemy’s health is reduced accordingly.
4. If health > 0, enemy continues behavior (e.g., attack, chase).
5. If health ≤ 0, trigger the **Die** event.

**Exceptions:**

* **Step 2 Exception:** Enemy has invincibility frames (post-hit immunity) → No damage is applied.

**Postconditions:** Enemy either continues fighting or dies if health reaches zero.  
**Priority:** 1 (must-have)  
**ID:** UC07

**Name:** Die **(<<include>> from Take Damage when health reaches zero)**

**Summary:** The enemy dies when its health reaches zero.  
**Actors:** Game Engine  
**Preconditions:** Enemy’s health is zero or below.  
**Basic Sequence:**

1. Enemy’s health reaches zero after taking damage.
2. Death animation is triggered.
3. Enemy is removed from active gameplay (despawned or left as a corpse sprite).
4. Possible drop of loot or points

**Exceptions:**

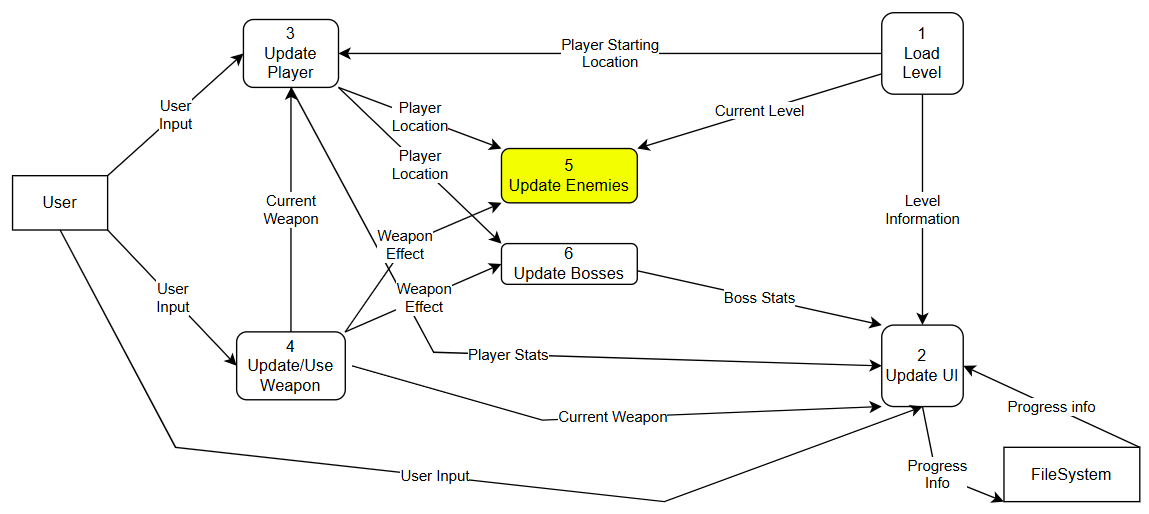
* **Step 3 Exception:** Enemy has a death-triggered ability (like an explosion) → Ability activates before despawning.

**Postconditions:** Enemy no longer affects gameplay after death.  
**Priority:** 1 (must-have)  
**ID:** UC08

## 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]

### Data Flow Diagrams



### Process Descriptions

**1. Enemy Movement Process**

* **Trigger:** Game loop updates (e.g., every frame).
* **Description:**
  1. **Check Activation Range:** Determine if the player is within the enemy’s activation range.
  2. **Decide Movement Type:**
     + IF player is outside detection range → Patrol (follow preset path).
     + IF player is within detection range → Chase player (move towards player).
  3. **Obstacle Detection:**
     + IF an obstacle is detected → Adjust movement path (jump, climb, or change direction).
  4. **Update Position:** Apply calculated movement vectors to the enemy’s position.
  5. **Loop:** Repeat steps as long as the enemy is active.

**2. Enemy Attack Process**

* **Trigger:** Player detected within attack range.
* **Description:**
  1. **Detection Phase:**
     + Monitor player’s position relative to enemy.
     + IF player enters attack range → Proceed to attack preparation.
  2. **Attack Preparation:**
     + Apply cooldown timer (prevent constant attacks).
     + Play attack animation wind-up (if applicable).
  3. **Execute Attack:**
     + Activate attack hitbox or trigger ranged projectile.
     + Check for collision with the player.
  4. **Post-Attack Cooldown:**
     + Apply a brief delay before the next attack can be initiated.
  5. **Reset:** Return to Detection Phase.

**3. Enemy Damage Handling Process**

* **Trigger:** Enemy is hit by a player’s attack.
* **Description:**
  1. **Collision Detection:**
     + Detect if enemy’s hitbox overlaps with an active player attack hitbox.
  2. **Calculate Damage:**
     + Apply damage based on player’s attack power.
  3. **Apply Effects:**
     + IF damage is taken → Reduce health, trigger hit animation.
     + IF health ≤ 0 → Initiate death sequence (remove enemy from scene, play death animation).
  4. **Temporary Invulnerability (Optional):**
     + Apply brief invulnerability window to prevent immediate repeated hits.
  5. **Loop:** Continue monitoring for further damage events.

**4. Exception Handling Process (Edge Cases)**

* **Trigger:** When abnormal behavior occurs (e.g., glitches or unexpected game states).
* **Description:**
  1. **Check for Invalid States:**
     + IF enemy moves outside of the map boundaries → Reset position to spawn point.
     + IF AI behavior loop gets stuck (e.g., running into a wall repeatedly) → Trigger recovery behavior (change direction).
  2. **Fail-Safe Mechanisms:**
     + IF enemy’s health drops below zero unexpectedly → Correct to zero and trigger death.
     + IF multiple conflicting AI states are triggered simultaneously → Prioritize highest-risk state (e.g., attack over idle).
  3. **Log Errors (Optional):**
     + Record errors for debugging (if error logging is implemented).

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

**1. Enemy Patrol**

**Test Case 1:** Basic Patrol Movement

* **Input:** Enemy spawns in patrol mode with a predefined path.
* **Expected Output:** Enemy moves along the path smoothly, reversing or looping at the path's end.

**Test Case 2:** Obstacle Detection (Boundary Case)

* **Input:** An obstacle is suddenly placed in the enemy's path.
* **Expected Output:** Enemy stops, recalculates its path, or waits if no alternative path exists.

**Test Case 3:** Patrol Interrupted by Player Detection (Exception)

* **Input:** Player enters the enemy’s detection range mid-patrol.
* **Expected Output:** Enemy immediately stops patrolling and transitions to the Chase Player state.

**2. Detect Player**

**Test Case 1:** Successful Detection

* **Input:** Player moves within the enemy’s detection range and is in direct line of sight.
* **Expected Output:** Enemy switches to chase mode.

**Test Case 2:** Detection Fails Due to Obstacle (Exception)

* **Input:** Player is within range but hidden behind an obstacle.
* **Expected Output:** Enemy does not detect the player and continues patrolling.

**Test Case 3:** Edge of Detection Range (Boundary Case)

* **Input:** Player is exactly at the maximum detection distance.
* **Expected Output:** Enemy detects the player accurately if within line of sight.

**3. Chase Player**

**Test Case 1:** Basic Chase

* **Input:** Player is detected and moves away from the enemy.
* **Expected Output:** Enemy follows the player, dynamically adjusting its path.

**Test Case 2:** Lose Player Trigger (Exception)

* **Input:** Player moves out of range or breaks the line of sight during the chase.
* **Expected Output:** Enemy continues chasing briefly, then triggers the Lose Player state.

**Test Case 3:** Close Range Transition to Attack (Boundary Case)

* **Input:** Player moves just within the enemy’s attack range during the chase.
* **Expected Output:** Enemy stops chasing and transitions to the Attack Player state.

**4. Lose Player**

**Test Case 1:** Player Lost After Chase

* **Input:** Player escapes out of detection range, breaking the line of sight.
* **Expected Output:** Enemy continues searching briefly, then returns to patrol mode.

**Test Case 2:** Player Reappears Before Timer Expires (Exception)

* **Input:** Player re-enters the enemy’s detection range before the search timer ends.
* **Expected Output:** Enemy resumes the chase immediately.

**Test Case 3:** Search Timer Edge Case (Boundary Case)

* **Input:** Player disappears exactly as the persistence timer expires.
* **Expected Output:** Enemy correctly transitions back to patrol mode without errors.

**5. Attack Player**

**Test Case 1:** Successful Attack

* **Input:** Enemy is within attack range, and the player remains stationary.
* **Expected Output:** Enemy attacks, the animation plays, and damage is dealt to the player.

**Test Case 2:** Missed Attack Due to Dodging (Exception)

* **Input:** Player dodges or moves out of the attack range mid-animation.
* **Expected Output:** Attack misses, triggering the Miss Attack event.

**Test Case 3:** Edge of Hitbox (Boundary Case)

* **Input:** Player is positioned exactly on the edge of the enemy’s attack hitbox.
* **Expected Output:** Attack connects if within the hitbox, otherwise misses.

**6. Miss Attack**

**Test Case 1:** Basic Miss Scenario

* **Input:** Enemy performs an attack, and the player dodges successfully.
* **Expected Output:** Enemy’s attack misses, with no damage dealt to the player.

**Test Case 2:** Player Counterattacks During Recovery (Exception)

* **Input:** Player attacks the enemy during the enemy’s post-miss recovery frames.
* **Expected Output:** Enemy takes damage before resuming its behavior.

**Test Case 3:** Rapid Dodge at the Last Frame (Boundary Case)

* **Input:** Player dodges at the very last animation frame before the attack connects.
* **Expected Output:** Enemy’s attack misses if the player exits the hitbox in time.

**7. Take Damage**

**Test Case 1:** Basic Damage Application

* **Input:** Player’s attack connects with the enemy.
* **Expected Output:** Enemy’s health decreases according to the damage formula.

**Test Case 2:** Invincibility Frames Active (Exception)

* **Input:** Player attacks during the enemy’s invincibility period.
* **Expected Output:** Enemy does not take damage.

**Test Case 3:** Health Exactly Reaches Zero (Boundary Case)

* **Input:** Player deals just enough damage to reduce the enemy’s health to exactly zero.
* **Expected Output:** Enemy triggers the Die event without health going negative.

**8. Die**

**Test Case 1:** Enemy Dies After Taking Damage

* **Input:** Enemy’s health reaches zero after being hit.
* **Expected Output:** Death animation plays, enemy is removed from gameplay, and loot drops (if applicable).

**Test Case 2:** Death-Triggered Ability (Exception)

* **Input:** Enemy with an on-death ability (like an explosion) dies.
* **Expected Output:** Ability activates immediately before the enemy despawns.

**Test Case 3:** Multiple Hits on Death Frame (Boundary Case)

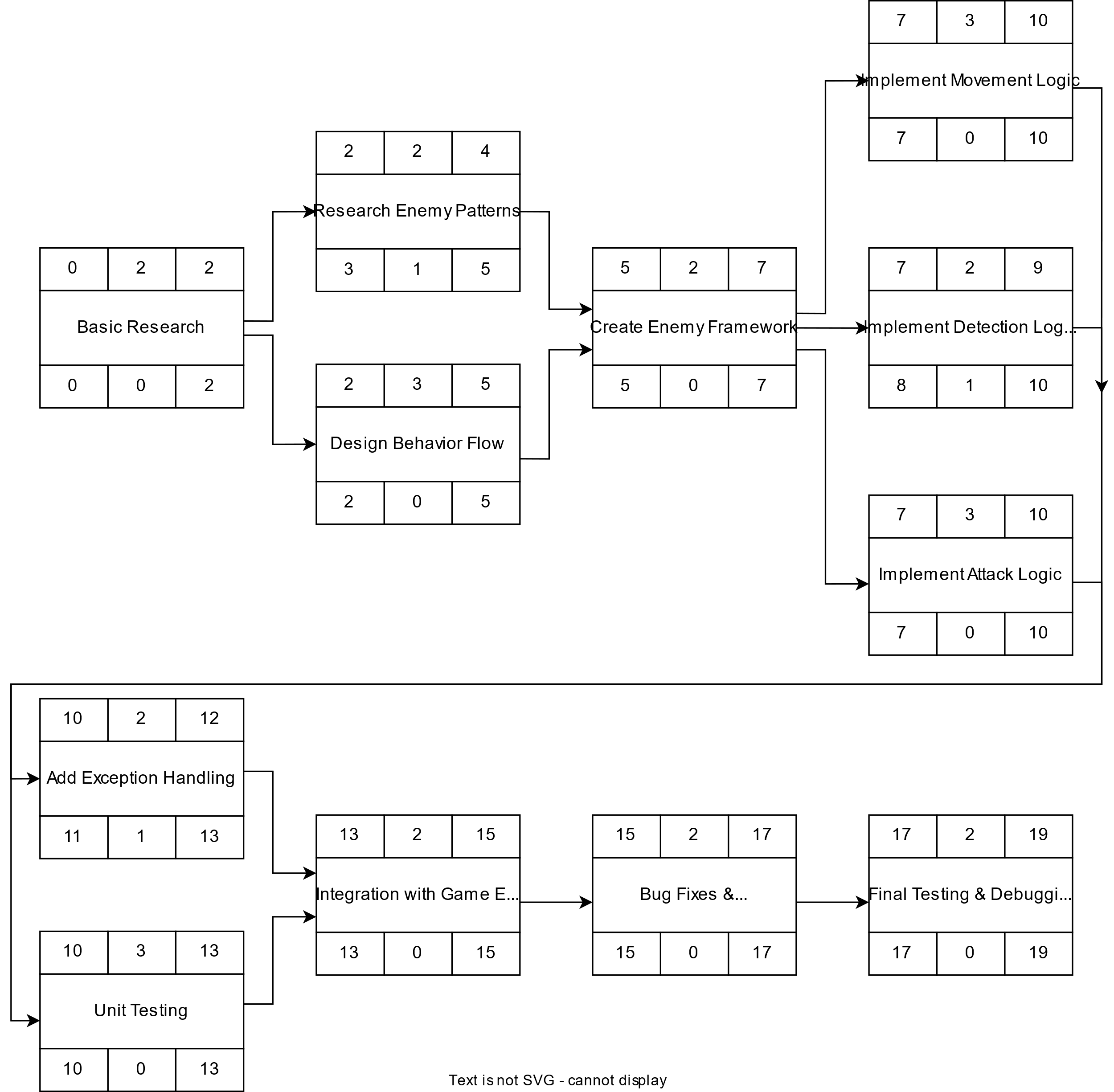
* **Input:** Multiple player attacks hit the enemy simultaneously as it dies.
* **Expected Output:** Enemy dies correctly, with no duplicate death animations or crashes.

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

### Work items

|  |  |  |
| --- | --- | --- |
| Task | Duration (hours) | Predecessor Task(s) |
| 1. Requirements Gathering | 2 | - |
| 2. Research Enemy AI Patterns | 2 | 1 |
| 3. Design Enemy Behavior Flow | 3 | 1 |
| 4. Create Basic Enemy Framework | 2 | 2, 3 |
| 5. Implement Movement Logic | 3 | 4 |
| 6. Implement Detection Logic | 2 | 4 |
| 7. Implement Attack Logic | 3 | 4 |
| 8. Add Exception Handling | 2 | 5, 6, 7 |
| 9. Unit Testing for Each Feature | 3 | 5, 6, 7 |
| 10. Integration with Game Engine | 2 | 8, 9 |
| 11. Bug Fixes & Optimization | 2 | 10 |
| 12. Final Testing & Debugging | 2 | 11 |

### Pert diagram



### Gantt timeline

