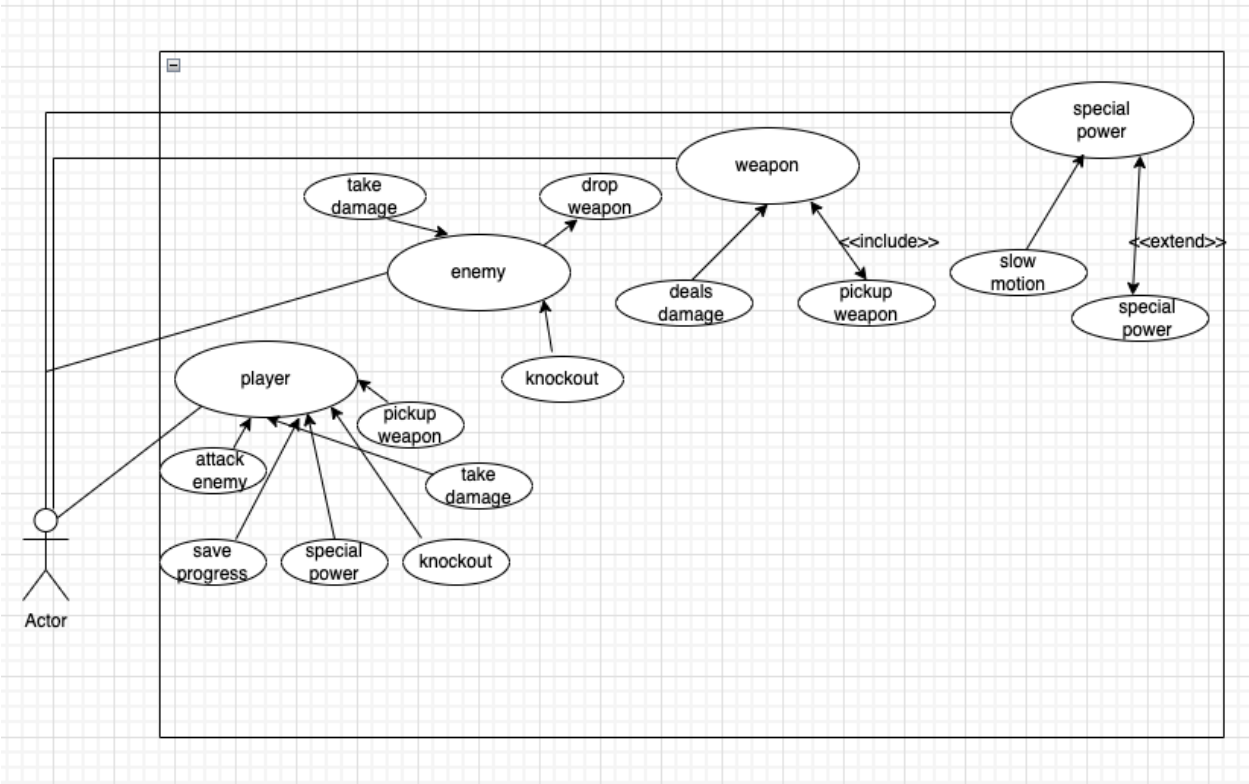
Name: Kushal khadka Date: 5th feb ,2025

Brief Introduction __/3

The 2D Combat System enhances Unity-based 2D gameplay with smooth sprite-based animations for attacks, knockouts, and weapon interactions. It integrates physics-based mechanics, including ragdoll physics for dynamic knockouts and realistic weapon handling for knives, swords, daggers, and hammers. Special powers trigger slow-motion effects for cinematic impact. Combining animation, physics, and visual effects, the system delivers an immersive and polished combat experience.

Use case diagram with scenario __14



Scenarios

Scenario 1: Attacks Enemy
Name: Attacks Enemy
Summary: The player attacks an enemy during combat.
Actors: Player, Enemy.
Preconditions: The player and enemy are in combat range.
Basic Sequence:
Step 1: Player presses the attack button.
Step 2: The attack animation is triggered.
Step 3: The enemy takes damage based on the weapon or attack type.
Step 4: The enemy's health is updated.
Exceptions:
Step 1: Player is out of range: Display "Out of Range" message.
Step 3: Enemy is already defeated: Ignore the attack.
Post Conditions: Enemy health is reduced, and the combat state is updated.
Priority: 1 (Must Have)
ID: UC01

Scenario 2: Picks Up Weapon
Name: Picks Up Weapon
Summary: The player picks up a weapon during their journey.
Actors: Player, Weapon.
Preconditions: A weapon is available in the game world.
Basic Sequence:
Step 1: Player moves near the weapon.
Step 2: Player presses the interact button.
Step 3: The weapon is equipped to the player.
Step 4: The weapon's stats are applied to the player's attacks.
Exceptions:
Step 2: Player is already holding a weapon: Display "Cannot carry more than one weapon" message.
Step 3: Weapon is not interactable: Ignore the input.
Post Conditions: The player is equipped with the weapon, and the weapon is removed from the game world.
Priority: 2 (Essential)
ID: UC02

Scenario 3: Activates Special Power
Name: Activates Special Power
Summary: The player activates a special power, triggering slow-motion effects.

Actors: Player.
Preconditions: The player has enough energy or mana to activate the special power.
Basic Sequence:
Step 1: Player presses the special power button.
Step 2: The slow-motion effect is activated.
Step 3: The game world slows down for a set duration.
Step 4: Normal time resumes after the duration ends.
Exceptions:
Step 1: Player does not have enough energy: Display "Not enough energy" message.
Step 2: Special power is on cooldown: Display "Power on cooldown" message.
Post Conditions: Slow-motion effect is applied, and the player's energy is reduced.
Priority: 2 (Essential)
ID: UC03

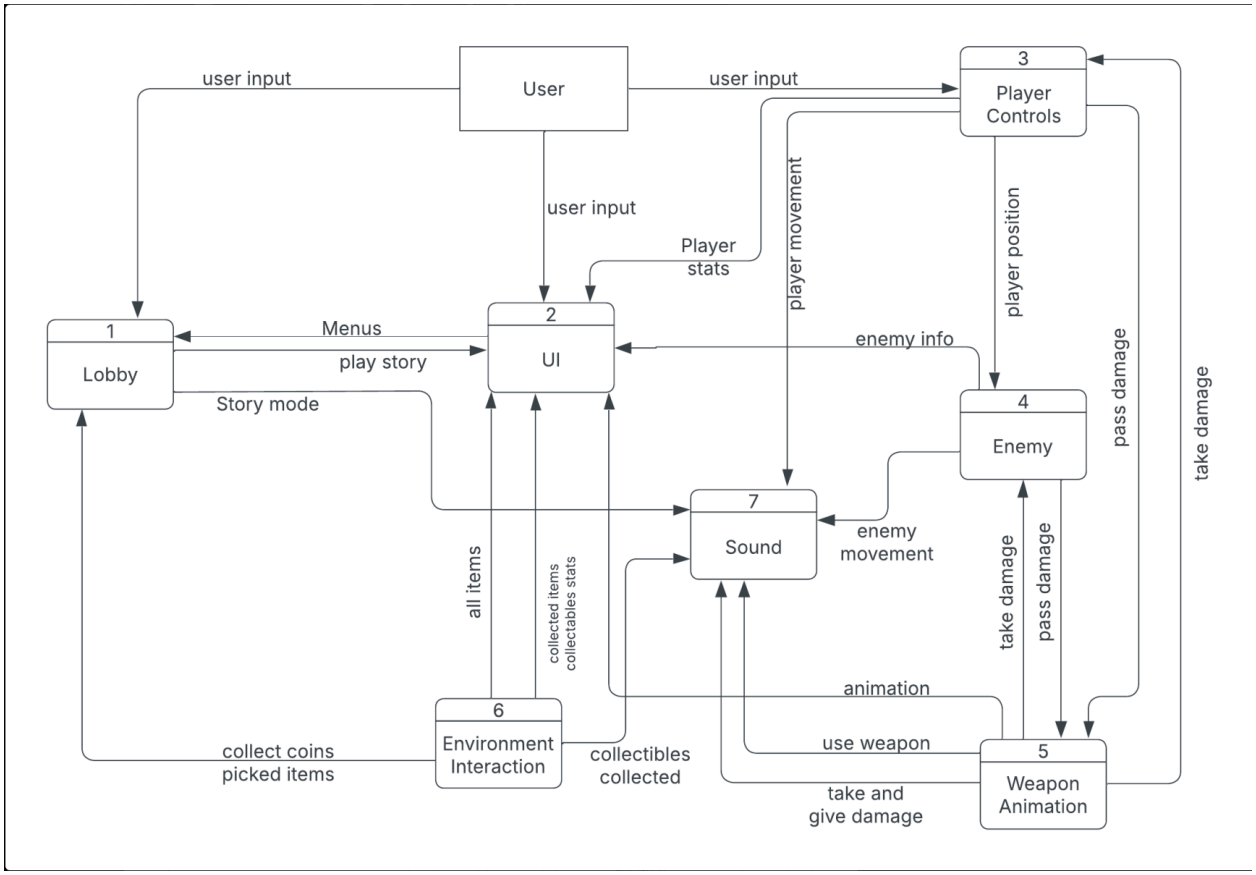
Scenario 4: Takes Damage
Name: Takes Damage
Summary: The player or enemy takes damage from an attack.
Actors: Player, Enemy.
Preconditions: The player or enemy is in combat.
Basic Sequence:
Step 1: An attack hits the player or enemy.
Step 2: Damage is calculated based on the weapon or attack type.
Step 3: Health is reduced by the damage amount.
Step 4: If health reaches zero, the knockout sequence is triggered.
Exceptions:
Step 1: Attack misses: Ignore damage calculation.
Step 3: Health is already zero: Ignore further damage.
Post Conditions: Health is updated, and knockout is triggered if applicable.
Priority: 1 (Must Have)
ID: UC04

Scenario 5: Knocked Out
Name: Knocked Out
Summary: The player or enemy is defeated and knocked out.
Actors: Player, Enemy.
Preconditions: The player or enemy's health reaches zero.

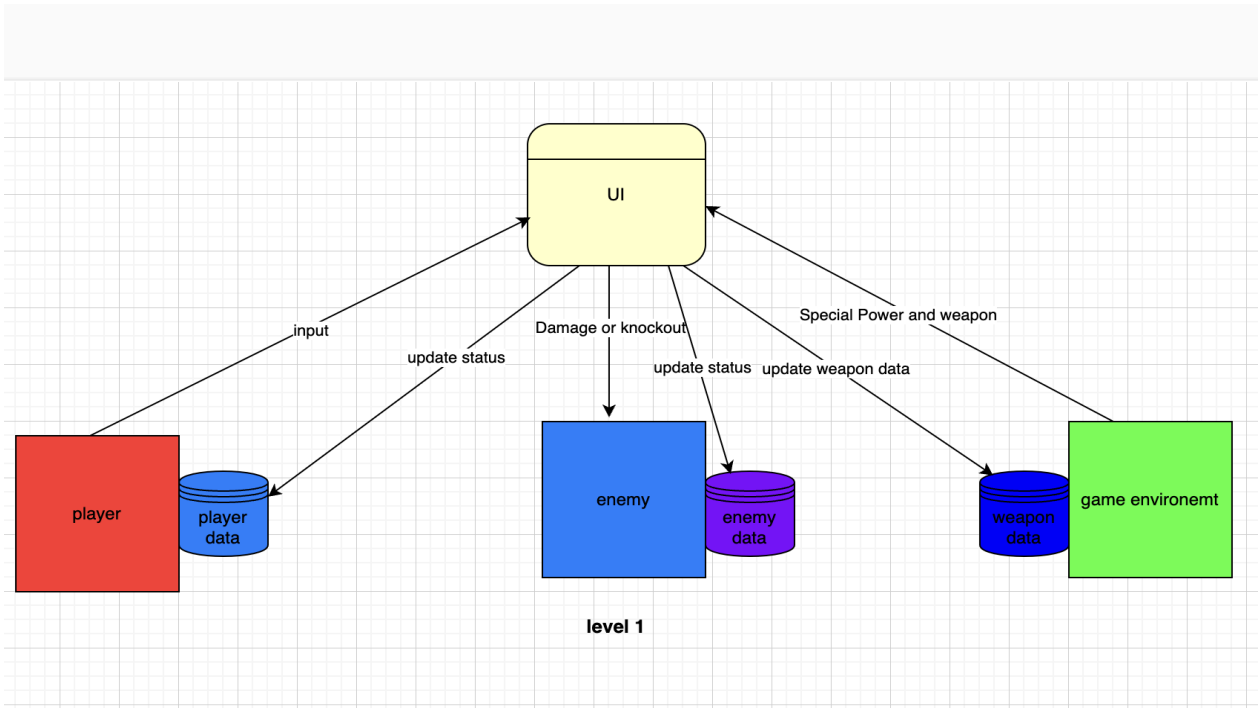
Basic Sequence:
Step 1: Health reaches zero.
Step 2: Knockout animation is triggered.
Step 3: Ragdoll physics is applied to the defeated character.
Step 4: The character is removed from combat.
Exceptions:
Step 2: Character is already in a knockout state: Ignore the sequence.
Step 3: Ragdoll physics fails: Fallback to a static knockout animation.
Post Conditions: The character is defeated and removed from the game world.
Priority: 1 (Must Have)
ID: UC05

Scenario 6: Saves Progress
Name: Saves Progress
Summary: The player saves their current game progress.
Actors: Player.
Preconditions: The game is in a saveable state (e.g., not in combat).
Basic Sequence:
Step 1: Player opens the menu and selects "Save Game."
Step 2: The game state (e.g., health, weapons, progress) is saved.
Step 3: A confirmation message is displayed.
Exceptions:
Step 1: Game is in combat: Display "Cannot save during combat" message.
Step 2: Save file is corrupted: Display "Save failed" message.
Post Conditions: The game progress is saved, and the player can resume later.
Priority: 3 (Nice to Have)
ID: UC06

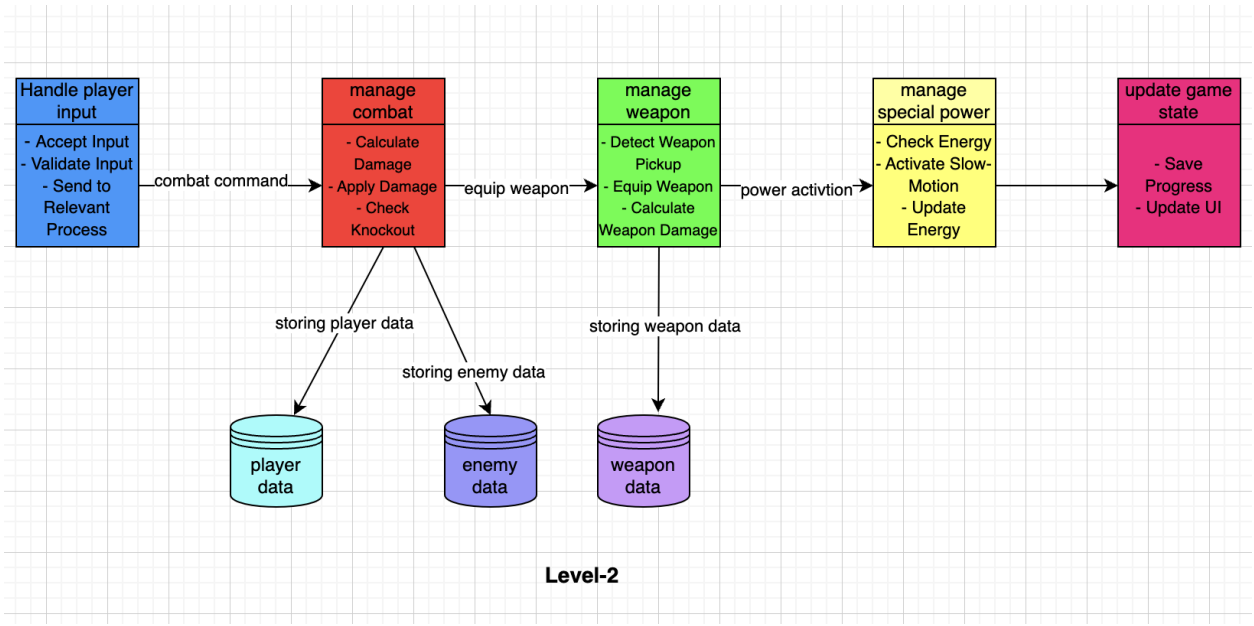
Data Flow diagram(s) from Level 0 to process description for your feature _____14



Level 0



level 1



Level-2

1. Handle Player Input

Description: This process accepts and validates player inputs, then routes them to the appropriate sub-process (Combat, Weapons, or Special Powers).

Pseudocode:

```
WHILE game is running:
   IF player presses ATTACK button:
      Send "Attack Input" to Manage Combat
   ELSE IF player presses INTERACT button:
      Send "Weapon Pickup Command" to Manage Weapons
   ELSE IF player presses SPECIAL POWER button:
      Send "Special Power Activation" to Manage Special Powers
   END IF
END WHILE
```

2. Manage Combat

Description: This process handles combat mechanics, including damage calculation, applying damage, and checking for knockouts.

Pseudocode:

```
WHILE combat is active:
   IF attack hits enemy:
      Calculate damage based on weapon or attack type
      Apply damage to enemy
      IF enemy health <= 0:
         Trigger knockout physics
         Update "Knockout Status" in Update Game State
      END IF
   END IF
   Update "Damage Data" in Update Game State
END WHILE
```

3. Manage Weapons

Description: This process handles weapon-related actions, such as detecting weapon pickup, equipping weapons, and calculating weapon damage.

Pseudocode:

```
WHILE player is near a weapon:
   IF player presses INTERACT button:
      IF player is not already holding a weapon:
         Equip weapon to player
         Update "Weapon Equip Data" in Update Game State
```

```
        ELSE:
            Display "Cannot carry more than one weapon"
        END IF
    END IF
END WHILE
```

4. Manage Special Powers

Description: This process handles special power activation, including energy checks and slow-motion effects.

Pseudocode:

```
WHILE special power is available:
    IF player presses SPECIAL POWER button:
        IF player has enough energy:
            Activate slow-motion effect
            Reduce player energy
            Update "Slow-Motion Status" in Update Game State
        ELSE:
            Display "Not enough energy"
        END IF
    END IF
END WHILE
```

5. Update Game State

Description: This process updates the game state, including saving progress and updating the UI.

Pseudocode:

```
WHILE game is running:
    IF "Damage Data" received from Manage Combat:
        Update player and enemy health in UI
    END IF
    IF "Weapon Equip Data" received from Manage Weapons:
        Update equipped weapon in UI
    END IF
    IF "Slow-Motion Status" received from Manage Special Powers:
        Update slow-motion effect in UI
    END IF
    Save progress to Player Data, Enemy Data, and Weapon Data
END WHILE
```

1. Handle Player Input

Objective: Ensure the system correctly processes player inputs and routes them to the appropriate sub-process.

Test case	Input	Expected output	Notes
Attacked button pressed	Player presses the attack button	“Attack input send to manage combat	Verify combat system receives the input.
Interact button pressed	Player presses interact button	"Weapon Pickup Command" sent to Manage Weapons	Verify weapon system receives the input
Special button pressed	Player presses the special power button	"Special Power Activation" sent to Manage Special Powers	Verify the special power system receives the input.
Invalid input	When player presses invalid button	Ignore input	Ensures no action is taken for invalid input

2. Manage Combat

Objective: Ensure the combat system correctly calculates damage, applies it, and handles knockouts.

Test case	Input	Expected output	Notes
Attack Hits enemy	Player attacks enemy	Enemy health reduced by weapon damage	Verify the damage calculation and health updates
Enemy health <= 0	Enemy health reaches 0	Knockout physics is triggered	Verify knockout animation and physic
Attack misses	Player misses the attack	No damage applied	Ensures no health damage in enemy
Boundary: max damage	Weapon deals max damage	Enemy health reduced to 0	Verify system handles maximum

			health damage correctly
Boundary: min damage	Weapon deals min damage	Enemy health reduced by 1	Verify system handles minimum health damage

3. Manage Weapons

Objective: Ensure the weapon system correctly handles weapon pickup, equipping, and damage calculation.

Test case	Input	Expected Output	Notes
Pickup Weapon	Player pickup weapon	Weapon equipped by the player	Verified weapons are added to the player's inventory.
Equip weapon	Player equip weapon	Weapon stats applied to player	Verify player's attack damage is updated.
Cannot Carry Multiple Weapons	Player tries to pick up a second weapon	Display "Cannot carry more than one weapon"	Ensure player’s can only carry one weapon at a time.
Boundary: Max Weapon Damage	Weapon deals max damage	Enemy health reduced to 0	Verify system handles maximum weapon damage values correctly
Boundary: Min Weapon Damage	Weapon deals min damage	Enemy health reduced by 1	Verify system handles minimum weapon damage values correctly.

4. Manage Special Powers

Objective: Ensure the special power system correctly handles activation, energy checks, and slow-motion effects.

Test case	Input	Expected Output	Notes
Activate Special Power	Player activates special power	Slow-motion effect triggered	Verify slow-motion is applied and energy is reduced.
Not Enough Energy	Player has insufficient energy	Display "Not enough energy	Ensure special power cannot be activated without sufficient energy.
Boundary: Max Energy	Player has max energy	Special power activated successfully	Verify system handles maximum energy values correctly.
Boundary: Min Energy	Player has min energy	Special power cannot be activated	Verify system handles minimum energy values correctly.

5. Update Game State

Objective: Ensure the game state is correctly updated and saved.

Test case	Input	Expected output	Notes
Update Player Health	Player takes damage	Player health updated in UI	Verify health bar reflects current health.
Update Enemy Health	Enemy takes damage	Enemy health updated in UI	Verify health bar reflects current health.
Update Equipped Weapon	Player equips weapon	Weapon icon updated in UI	Verify UI reflects the currently equipped weapon
Save Progress	Player saves game	Progress saved to Player Data, Enemy Data, Weapon Data	Verify data is correctly saved and can be loaded.
Boundary: Max Health	Player health reaches max	Health bar displays max value	Verify system handles maximum health values correctly.
Boundary: Min Health	Player health reaches 0	Health bar displays 0, knockout triggered	Verify system handles minimum health values correctly.

Example Test Case Execution

Test Case: Attack Hits Enemy

1. **Input:** Player presses ATTACK button.
2. **Steps:**
    - System receives "Attack Input".
    - System calculates damage based on weapon stats.
    - System applies damage to enemy.
    - System updates enemy health in UI.
3. **Expected Output:**
    - Enemy health is reduced by the correct amount.
    - If enemy health <= 0, knockout physics is triggered.

Test Case: Pick Up Weapon

1. Input: Player presses INTERACT button near a weapon.
2. **Steps:**
    - System receives "Weapon Pickup Command".
    - System checks if player is already holding a weapon.
    - If not, system equips the weapon to the player.
    - System updates weapon stats in UI.
3. **Expected Output:**
    - Weapon is equipped to player.
    - Player's attack damage is updated.
    - Weapon icon is displayed in UI.

Boundary Cases

1. **Max Damage:**
    - Input: Weapon deals max damage.
    - Expected Output: Enemy health reduced to 0, knockout triggered.
2. **Min Damage:**
    - Input: Weapon deals min damage.
    - Expected Output: Enemy health reduced by 1.
3. **Max Energy:**
    - Input: Player has max energy.
    - Expected Output: Special power activated successfully.
4. **Min Energy:**
    - Input: Player has min energy.
    - Expected Output: Special power cannot be activated.

5. Timeline _____/10

Project Timeline

Assumptions:

- Start Date: March 1
- End Date: April 30
- Total Duration: 9 Weeks

1. Requirements Collection

- Duration: 1 Week
- Description: Gather and document all requirements for the 2D Combat System, including player inputs, combat mechanics, weapon systems, and special powers.
- Deliverables: Requirements document.

2. System Design (DFD, Use Cases)

- Duration: 1 Week
- Predecessor: Task 1
- Description: Create Data Flow Diagrams (DFDs) and Use Case Diagrams to design the system architecture.
- Deliverables: DFDs, Use Case Diagrams.

3. Sprite and Animation Creation

- Duration: 2 Weeks
- Predecessor: Task 2
- Description: Design and create sprites and animations for the player, enemies, and weapons.
- Deliverables: Sprite sheets, animations.

4. Physics Implementation

- Duration: 2 Weeks
- Predecessor: Task 2
- Description: Implement physics for knockouts, weapon interactions, and slow-motion effects.
- Deliverables: Physics system.

5. Weapon System Development

- Duration: 2 Weeks
- Predecessor: Task 2
- Description: Develop the weapon system, including weapon pickup, equipping, and damage calculation.
- Deliverables: Weapon system.

6. Special Power Implementation

- Duration: 2 Weeks
- Predecessor: Task 2
- Description: Implement special powers, including energy checks and slow-motion effects.
- Deliverables: Special power system.

7. Combat System Integration

- Duration: 2 Weeks
- Predecessor: Tasks 3, 4, 5, 6
- Description: Integrate all components (sprites, physics, weapons, special powers) into the combat system.
- Deliverables: Integrated combat system.

8. UI and Game State Updates

- Duration: 1 Week
- Predecessor: Task 7
- Description: Implement the UI for health bars, weapon icons, and special power indicators. Update the game state (e.g., saving progress).
- Deliverables: UI, game state management.

9. Testing and Debugging

- Duration: 1 Week
- Predecessor: Task 8
- Description: Test the system for bugs, performance issues, and edge cases. Fix any issues found.
- Deliverables: Tested and debugged system.

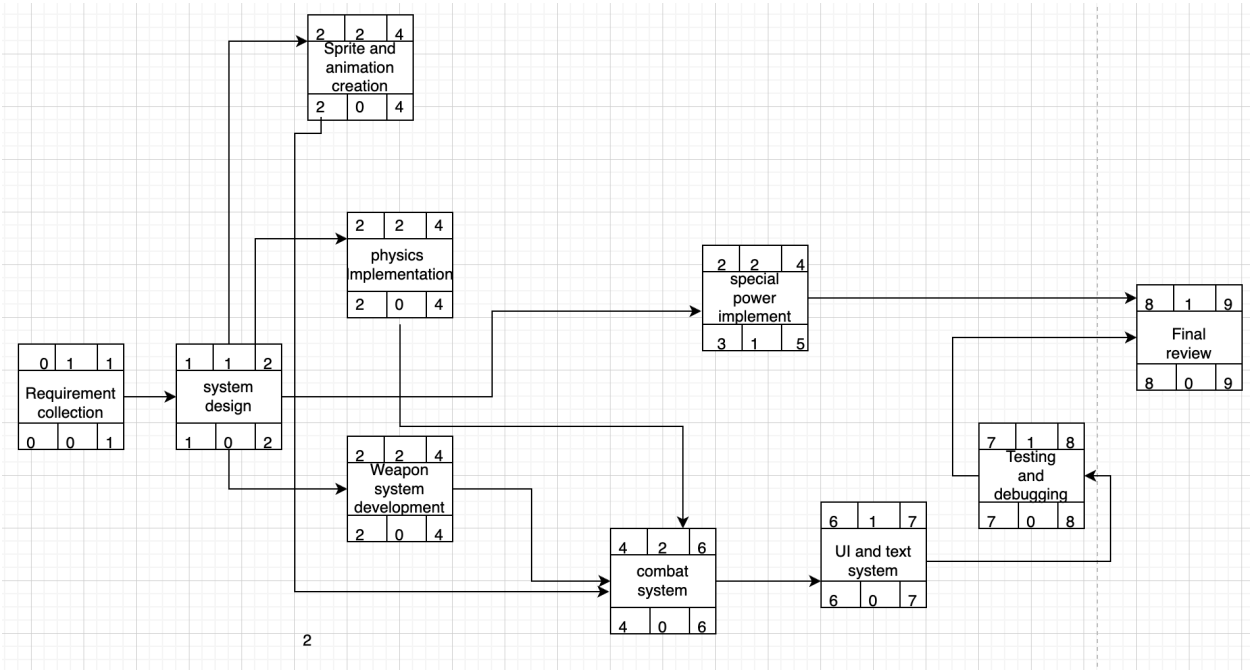
10. Final Review and Deployment

- Duration: 1 Week
- Predecessor: Task 9
- Description: Conduct a final review of the system and deploy it.
- Deliverables: Final deployed system.

Task	Duration (weeks)	Predecessor Tasks	Start date	End data
Requirement collection	1	-	March 1st	March 7th
System Design (DFD, Use Cases)	1	1	March 8th	March 14th
Sprite and Animation Creation	2	2	March 15th	March 28th
Physics	2	2	March 15th	March 28th

implementation				
Weapon system development	2	2	March 15th	March 28th
Special power Implementation	2	2	March 15th	March 28th
Combat system	2	3,4,5,6	March 15th	April 11th
UI and game system update	1	7	April 12th	April 18th
Testing and Debugging	1	8	April 19th	April 25th
Final Review and Deployment	1	9	April 26th	April 30

Pert diagram


Requirement collection
system design
Sprite and animation creation
physics implementation
Weapon system development
special power implement
combat system
UI and text system
Testing and debugging
Final review

Gantt timeline

	march				april				
	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	week 9
Requirement collection									
System Design (DFD, Use Cases)		1							
Sprite and Animation Creation			2						
Physics implementation				2					
Weapon system development				2					
Special power Implementation				2					
Combat system						3,4,5			
UI and game system update							7		
Testing and Debugging								8	
Final Review and Deployment									9