Name: Jaskaran Singh Mark _____/50

[**Instructions**: Remove everything that is not a heading below and fill in with your own diagrams, etc.]

1. Brief introduction _/3

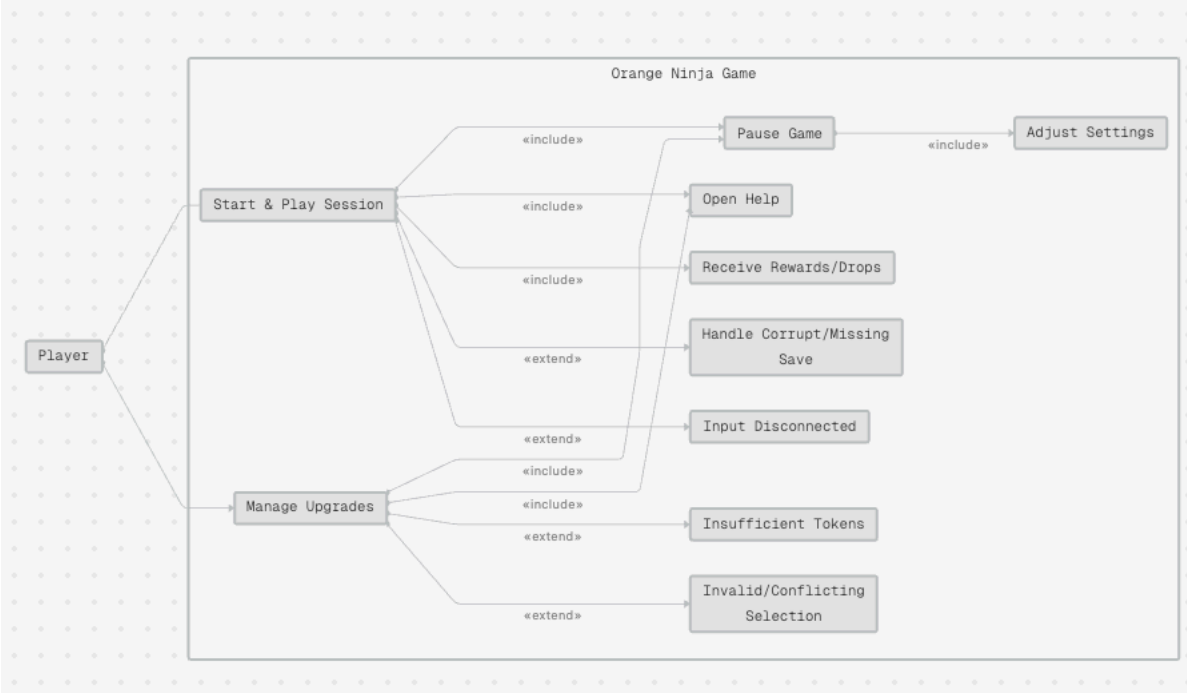
My feature for the Orange Ninja video game is to define the intended users and explain how they will interact with the system.

The goal of this feature is to identify who will actually be playing the game and outline their basic needs. The game is meant for players who enjoy rogue-lite action but want something lighter and more playful compared to the darker tone of many games in the genre.

I also need to describe how these users interact with the system at a simple level. For example, players control the Orange Ninja as they move through levels, fight enemies, and collect items. They also use menus to save progress, manage upgrades, or change settings. These interactions form the core of how the intended users experience the game.

2. Use case diagram with scenario _14

Use Case Diagrams



Scenarios

Scenario 1 (1st Use Case Diagram)

Name: Start & Play Session
Summary: The player starts the game, chooses a session, and plays a level using core controls.
Actors: Player
Preconditions: Game launches successfully; input device detected.

Basic sequence:
Step 1: Player reaches the **Main Menu**.
Step 2: Player selects **New Game** or **Load Game**.
Step 3 (include): Player may open **Help** to view controls/goals. <<include>> Open Help
Step 4: Level loads; player **moves/attacks** and **collects drops**.
Step 5 (include): Player may **Pause Game** at any time. <<include>> Pause Game
Step 6 (include): From Pause, player may **Adjust Settings** and resume play. <<include>> Adjust Settings
Step 7 (include): During play, the system grants **Rewards/Drops** for kills and objectives. <<include>> Receive Rewards/Drops

Exceptions (must map to sequence steps):
Step 2: Save file is **missing or corrupted** → run recovery and default to **New Game**; no crash. <<extend>> Handle Corrupt/Missing Save
Step 4: Input device disconnects during play → automatically pause and show "Reconnect device"; resume when reconnected. <<extend>> Input Disconnected

Post conditions: Player is in an active run with progress saved at checkpoints, or back at menu with settings preserved.
Priority: 1*
ID: PS1
*Priorities: 1 = must have, 2 = essential, 3 = nice to have.

Scenario 2 (2nd Use Case Diagram)

Name: Manage Upgrades
Summary: The player opens the upgrade menu and applies an upgrade to the Orange Ninja.
Actors: Player
Preconditions: Player is at an upgrade point or has upgrade tokens/currency.

Basic sequence:
Step 1 (include): Player opens **Pause Game** to access menus. <<include>> Pause Game
Step 2: Player opens the **Upgrades** screen.
Step 3: Player **browses** categories (weapons, stats, perks).
Step 4: Player **selects** an upgrade and **confirms** purchase.

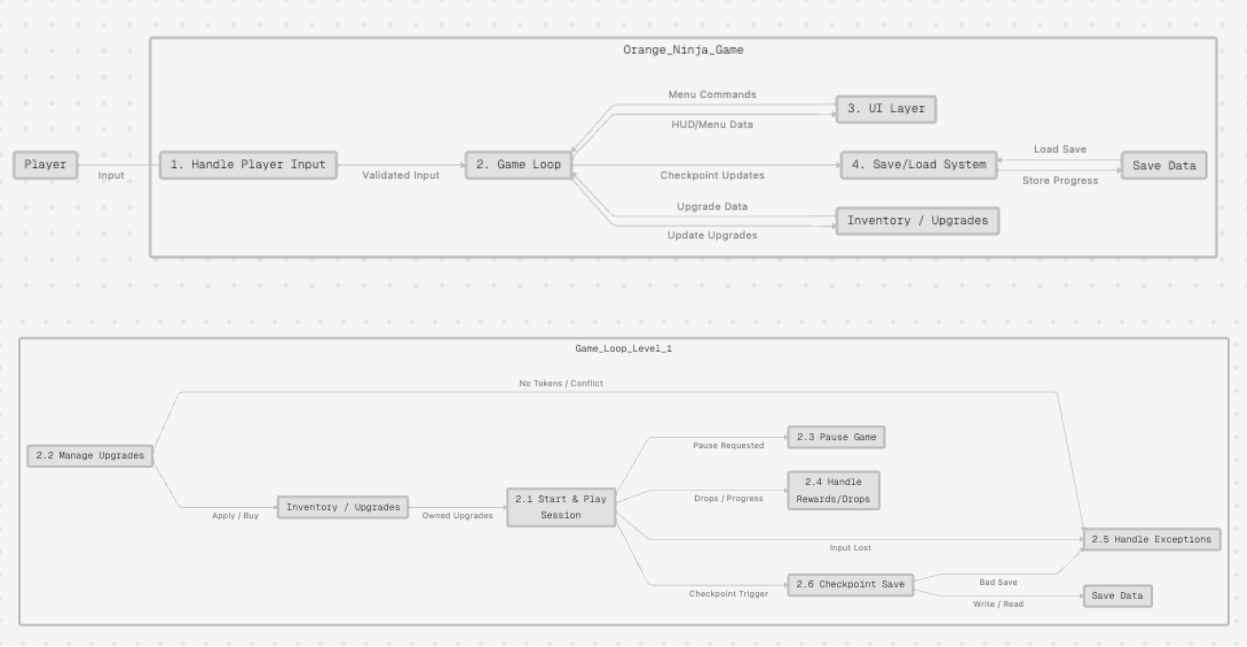
Step 5: System **applies** upgrade, updates **HUD/stats**, and **saves** state.
Step 6 (include): Player may open **Help** for upgrade hints/legend. <<include>> Open Help

Exceptions (must map to sequence steps):
Step 4: Player **lacks tokens** for the selected upgrade → show "Insufficient tokens," block purchase, remain in menu. <<extend>> Insufficient Tokens
Step 4: Player selects an **invalid or conflicting** upgrade → show reason (e.g., mutually exclusive), make no changes. <<extend>> Invalid/Conflicting Selection

Post conditions: Valid upgrade is applied and saved; otherwise, the attempted selection is rejected with no state change.
Priority: 2*
ID: UG1

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

Data Flow Diagrams



Process Descriptions

```
WHILE game session is active
  IF player provides input
    Validate input
    Apply action (movement, attack, item use)
  ENDIF

  IF player pauses game
    Display pause menu
    IF player selects settings/help
      Open corresponding screen
    ELSE IF player selects quit
      Save progress and exit to menu
    ENDIF
  ENDIF

  IF upgrade menu is opened
    Display available upgrades and token balance
    IF player has enough tokens AND upgrade is valid
      Deduct tokens
      Apply upgrade
      Save progress
    ELSE
      Reject selection with message (insufficient tokens or invalid choice)
    ENDIF
  ENDIF

  IF enemy defeated or milestone reached
    Generate rewards/drops
    Add to player inventory
    Save checkpoint
  ENDIF

  IF input device disconnects
    Auto-pause game
    Wait until device reconnects
  ENDIF
END WHILE
```

4. Acceptance Tests _____9

This feature involves both **deterministic elements** (game actions such as pausing, opening help, and managing upgrades) and **random elements** (item drops, reward generation, and token usage). Acceptance tests will ensure that:

- The system correctly executes core player actions (Start/Play Session, Manage Upgrades).
- Boundary cases are handled properly (e.g., insufficient tokens, corrupted save, disconnected input).
- Random outcomes such as drops or rewards remain within acceptable bounds and never exceed defined constraints.

The Acceptance tests for these features are described below.

Gameplay Session Test

- **Description:** Verify that the player can successfully start, pause, and resume a play session.
- **Steps:**
    1. Start new game session.
    2. Trigger pause → verify settings/help menu opens.
    3. Resume session → verify state resumes from correct point.
- **Expected Output:** Game resumes without data loss.
- **Failure Case:** If pause/resume causes reset, or game state (HP, items) is lost.

Upgrade System Test

- **Description:** Verify upgrades can be applied only if tokens are sufficient.
- **Steps:**
    1. Attempt upgrade with enough tokens.
    2. Attempt upgrade with insufficient tokens.
    3. Attempt multiple conflicting upgrades.
- **Expected Output:**

1. Valid upgrades apply successfully.
2. Insufficient tokens → error message.
3. Conflicting upgrades → rejected.

Reward & Drop Test

- **Description:** Verify reward distribution when defeating enemies/bosses.
- **Steps:**
    1. Defeat enemy 1000 times.
    2. Log frequency of each drop.
    3. Check boundary cases (rare drop rate, no duplicate drop beyond allowed).
- **Expected Output:**
    1. Common items appear >50 times.
    2. Rare items <5% probability.
    3. No item exceeds defined cap.

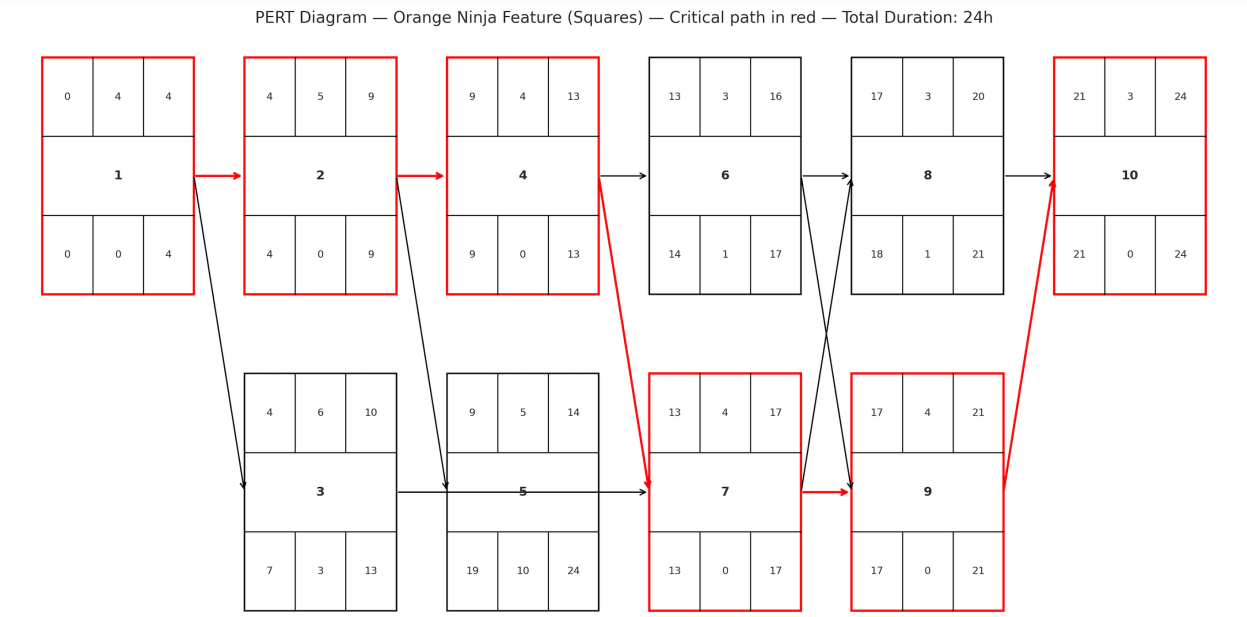
Test Case	Condition	Input / Action	Expected Result	Success	Fail
Pause/ Resume	Player pauses mid-level	Press Pause → Resume	Game resumes at same state	Yes	No
Insufficient Tokens	Attempt upgrade with 0 tokens	Select upgrade option	Error message shown, upgrade blocked	Yes	No
Conflicting Upgrade	Apply Speed+ and Speed- together	Select both	System rejects invalid combo	Yes	No
Reward Drop Rate	Defeat 1000 enemies	Logged output	Common items appear 50+ times, rare <5%	Yes	No
Save File Corruption	Load corrupted save file	Attempt load	Error handled, safe fallback	Yes	No

5. Timeline _____/10

Work items

Task	Duration (PWks)	Predecessor Task(s)
1. Feature Requirements Collection	4	-
2. Game Session Flow Design	5	1
3. Upgrade System Design	6	1
4. Token Validation Programming	4	2
5. Pause/Menu System Programming	5	2
6. Insufficient Token Handling	3	4
7. Upgrade Conflict Checking	4	3, 4
8. Documentation	3	6, 7
9. Testing	4	6, 7
10. Installation & Integration	3	8, 9

Pert diagram



Gantt timeline