Name: Ankit Paudel Mark \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/50

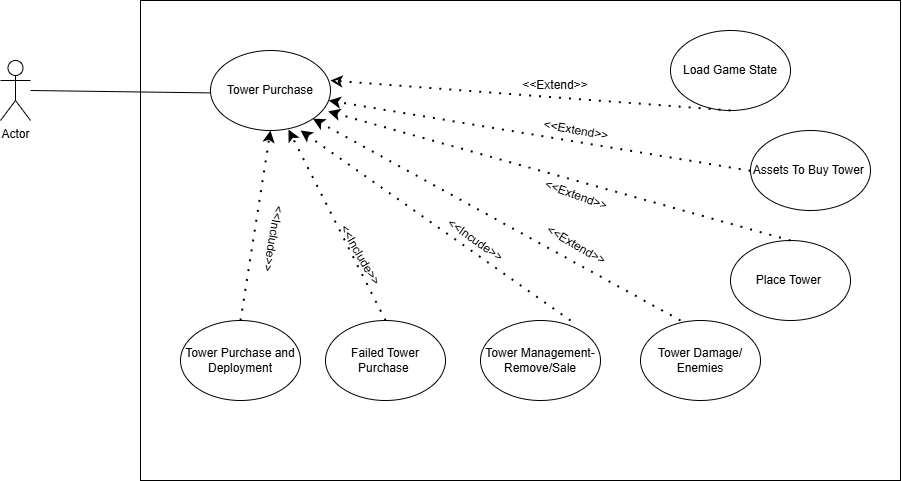
## Brief introduction \_\_/3

In "Kill Mo' Chickens," players engage in strategic tower defense gameplay, inspired by the popular Balloons Tower Defense series.

My core function in the game is purchase\_tower(), allows players to buy defense towers using in-game resources, crucial for planning and resource management. This function is integral to the game's systems, such as balancing resources and managing game state to ensure smooth gameplay. Players can place towers strategically on the map to form effective defense lines against incoming enemies. Each tower has specific enemy tracking and shooting mechanics, with damage types varied by enemy attributes, adding depth to combat strategies. The option to sell or remove towers provides tactical flexibility, allowing players to adapt their strategies and recover resources dynamically. This blend of strategic decision-making and responsive mechanics creates a thrilling, engaging gameplay experience where each decision significantly impacts progress and success.

## Use case diagram with scenario \_\_14

### Use Case Diagrams



### Scenarios

**Scenario 1: Standard Tower Purchase and Deployment**  
**Name:** Purchase and Deploy Tower  
**Summary:** The player selects, purchases, and deploys a tower on the game map to enhance defenses against incoming enemies.  
**Actors:** Player  
**Preconditions:**

* The game is in an active state.
* The player has accessed the tower purchase menu.  
  **Basic Sequence:**
* **Step 1:** Player selects a tower that fits their strategic needs and available resources.
* **Step 2:** System checks if the player has enough resources to complete the purchase.
* **Step 3:** Resources are deducted from the player’s total if sufficient.
* **Step 4:** Player enters placement mode and positions the tower on the map.
* **Step 5:** System validates the chosen location against game rules.
* **Step 6:** Upon successful validation, the tower is placed and activated, immediately beginning to track and engage nearby enemies.  
  **Exceptions:**
* **Step 2:** If resources are insufficient, the transaction is canceled and a hint is displayed to gain more resources.
* **Step 5:** If placement is invalid, feedback is provided to suggest valid placement options.  
  **Postconditions:**
* Game state reflects the new tower's presence, interacting with enemies as programmed.  
  **Priority:** 1  
  **ID:** TP1

**Scenario 2: Failed Tower Purchase**  
**Name:** Attempt Tower Purchase with Insufficient Funds  
**Summary:** The player attempts to purchase a tower but fails due to not having enough resources, receiving guidance on resource management.  
**Actors:** Player  
**Preconditions:**

* Player has selected a tower in the purchase menu.
* The game is in an active state.  
  **Basic Sequence:**
* **Step 1:** Player attempts to purchase a selected tower.
* **Step 2:** System checks and indicates a failure in resource verification due to insufficient funds.
* **Step 3:** Player receives a notification detailing the resource shortfall and suggestions for better resource management.  
  **Exceptions:** N/A  
  **Postconditions:**
* Player’s resource level remains unchanged.
* No new tower is placed on the map.  
  **Priority:** 1  
  **ID:** TP2

**Scenario 3: Efficient Tower Management - Removal/Sale**  
**Name:** Manage Tower Inventory by Removal/Sale  
**Summary:** The player sells or removes a tower to adapt their strategy, regain resources, or optimize tower placement.  
**Actors:** Player  
**Preconditions:**

* At least one tower is deployed on the map.
* The game is in an active state.  
  **Basic Sequence:**
* **Step 1:** Player selects an existing tower for removal or sale.
* **Step 2:** System displays removal or sale options, including potential resource recovery.
* **Step 3:** Player confirms the choice, and the tower is removed from the map.
* **Step 4:** Resources are added back to the player's total based on the tower’s sale value.  
  **Exceptions:**
* **Step 3:** If the tower was engaged in targeting or damaging enemies, the player may receive a warning or need to confirm the removal during combat.  
  **Postconditions:**
* The map is updated to reflect the tower's removal.
* The player’s resources are adjusted accordingly.  
  **Priority:** 1  
  **ID:** TP3

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

## Data Flow diagram(s) from Level 0 to process description for your feature \_\_\_\_\_\_\_14

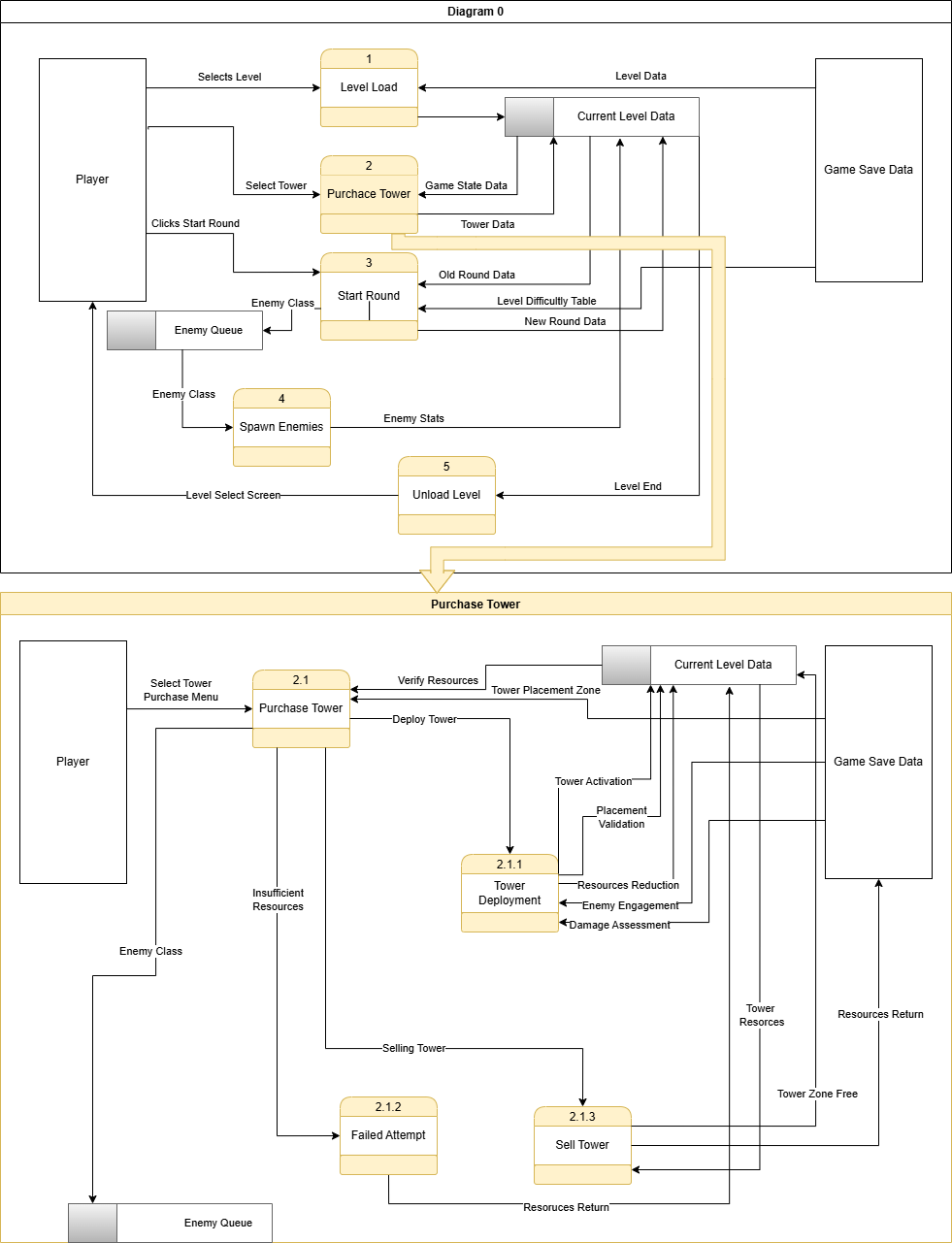
**System:** Tower System **Key Features:**

* **Resource Management:** Tracks and updates player resources.
* **User Input Handling:** Processes player interactions for tower management.
* **Validation Processes:** Ensures adherence to game rules before tower placement.

**Process Breakdown:**

1. **Tower Selection:** Players interact with the tower purchase interface to select towers based on strategy and resources.
2. **Resource Verification:** System checks if player resources are sufficient for the selected tower.
3. **Resource Deduction:** Upon successful verification, the tower cost is deducted from player's resources.
4. **Placement Mode Activation:** Players enter a mode to place the tower on the game map.
5. **Placement Validation:** The chosen location is validated against game rules.
6. **Tower Activation:** Validated towers are placed and become operational.
7. **Enemy Engagement:** Active towers automatically track and attack nearby enemies.
8. **Damage Assessment:** Towers assess and inflict damage based on their capabilities.
9. **Tower Adjustment:** Players may adjust or change towers based on game dynamics.
10. **Tower Removal/Sell:** Towers can be removed or sold, recouping some resources.

### Data Flow Diagrams



### Process Descriptions

**Tower Management System**:

1. **Initialize Tower Management**:
   * **Condition**: On player initiating tower placement or adjustment.
   * **Action**: Begin the tower management process.
2. **Tower Placement and Management Cycle**:
   * **WHILE** there are actions pending (select, place, adjust, sell towers):
     + **Tower Selection**:
       - **Input**: Player clicks on tower purchase interface.
       - **Process**: Display available towers based on current resources and strategy.
       - **Output**: Player selects a tower.
     + **Resource Verification**:
       - **Input**: Selected tower and its cost.
       - **Process**: Check if player's resources are sufficient.
       - **Output**: Approval or denial of tower purchase.
     + **IF** resources sufficient **THEN**:
       - **Resource Deduction**:
         * **Input**: Approved tower purchase.
         * **Process**: Deduct cost from player's resources.
         * **Output**: Update player's resource balance.
     + **Placement Mode Activation**:
       - **Input**: Tower purchase confirmed.
       - **Process**: Enable placement mode, allowing player to choose location.
       - **Output**: Player places the tower.
     + **Placement Validation**:
       - **Input**: Player's chosen location for the tower.
       - **Process**: Validate the chosen location against game rules.
       - **Output**: Confirm or reject placement.
     + **IF** placement valid **THEN**:
       - **Tower Activation**:
         * **Input**: Valid placement location.
         * **Process**: Activate tower at chosen location.
         * **Output**: Tower becomes operational.
       - **Enemy Engagement**:
         * **Input**: Operational tower.
         * **Process**: Tower automatically engages enemies within range.
         * **Output**: Interaction and damage to enemies.
       - **Damage Assessment**:
         * **Input**: Engagement data.
         * **Process**: Calculate damage dealt based on tower capabilities and enemy attributes.
         * **Output**: Update enemy health status.
     + **Tower Adjustment**:
       - **Input**: Player's strategic evaluation.
       - **Process**: Adjust tower placement or type based on gameplay dynamics.
       - **Output**: Apply changes to the tower setup.
     + **Tower Removal/Sell**:
       - **Input**: Player decision to modify strategy.
       - **Process**: Remove or sell the tower, recoup some resources.
       - **Output**: Update map and resource balance.
     + **Check for Further Actions**: Determine if more tower actions are pending.
   * **END WHILE**
3. **Finalize Tower Management**:
   * **IF** no more actions are pending:
     + **Confirm Tower Configurations**: Lock in all tower placements and adjustments.
     + **Signal Completion**: Notify the game engine that tower management phase is complete.
4. **End Tower Management Process**:
   * **Condition**: All towers are placed and managed according to player strategy.
   * **Action**: Conclude the tower management procedure.

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

The acceptance tests for the "Kill Mo' Chickens" game, specifically focusing on the purchase\_tower() function which is central to the gameplay. This tests will cover various boundary cases, ensuring that the tower purchase, deployment, and resource management features function as expected under different conditions.

**Acceptance Tests for purchase\_tower() Function**

| **Output** | **Resources Before** | **Resources After** | **Tower Cost** | **Notes** |
| --- | --- | --- | --- | --- |
| Success | 100 | 50 | 50 | Correctly processes transaction with exact resources. |
| Failure | 49 | 49 | 50 | Fails as expected due to insufficient resources; resources unchanged. |
| Success | 150 | 100 | 50 | Successfully purchases and updates resource balance. |
| Failure | 0 | 0 | 10 | Fails with zero resources; ensures no negative balance occurs. |
| Success | 1000 | 950 | 50 | Tests with high initial resources to ensure proper subtraction. |
| Failure | 30 | 30 | 100 | Checks that no deduction occurs when resources are insufficient. |
| Boundary | 50 | 0 | 50 | Boundary test where player just has enough resources. |

**Additional Considerations**

* **Placement Validation**: Ensure towers can only be placed in designated areas.
  + Inputs: Various coordinates on the game map.
  + Outputs: Confirmation or rejection of placement based on game rules.
* **Multiple Purchases**: Test rapid successive purchases and placements.
  + Input Sequence: Several purchase commands with sufficient resources.
  + Output: Each command processed correctly and resources updated accordingly.
* **Simultaneous Actions**: Attempt to purchase and sell towers simultaneously.
  + Input: Commands to buy and sell towers at the same time.
  + Output: Ensure game correctly handles simultaneous resource updates.
* **Edge Resource Cases**: Test right at the edge of having enough/not enough resources.
  + Input: Resource levels set at exact cost minus one, cost, and cost plus one.
  + Output: Verify proper handling of resource checks and messages displayed to the player.

These tests will help ensure that your purchase\_tower() function behaves correctly under a wide range of normal and extreme conditions, providing both robustness and a smooth user experience in "Kill Mo' Chickens."

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

### Work items

|  |  |  |
| --- | --- | --- |
| Task | Duration (PHrs) | Predecessor Task(s) |
| 1. Requirements Collection | 6 | - |
| 2. Tower Integration System | 8 | 1 |
| 3. Tower Deployment | 6 | 2 |
| 4. Tower Database Construction | 8 | 2,3 |
| 5. User Documentation | 4 | 4 |
| 6. Programming | 10 | 3 |
| 7. Testing | 6 | 6 |
| 8. Installation | 4 | 5, 7 |

### Pert diagram

### 

### Gantt timeline

