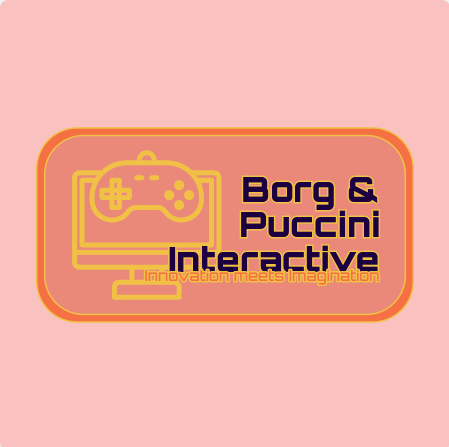
**Borg & Puccini Interactive**

**Design Document for:**

# Starve Wars

Welcome to Starve Wars: Outlast, outmaneuver, and out-eat your enemies in the ultimate food fight!

****

All work Copyright ©2024 by Borg Puccini Interactive

Written by Alissa Borg and Michael Puccini

Version # 1.00 Sunday, October 13, 2024Table of Contents

Table of Contents

[Starve Wars 1](#_Toc179735127)

[Game Overview 3](#_Toc179735128)

[Common Questions 3](#_Toc179735129)

[What is the game? 3](#_Toc179735130)

[What is the target audience? 3](#_Toc179735131)

[Why create this game? 3](#_Toc179735132)

[Where does the game take place? 4](#_Toc179735133)

[What do I control? 4](#_Toc179735134)

[How many characters do I control? 4](#_Toc179735135)

[What is the main focus? 4](#_Toc179735136)

[What’s different? 4](#_Toc179735137)

[Team Roles 5](#_Toc179735138)

[Feature Set 5](#_Toc179735139)

[General Features 5](#_Toc179735140)

[Gameplay 5](#_Toc179735141)

[Flowchart 7](#_Toc179735142)

[Storyboards 7](#_Toc179735143)

[Controls 8](#_Toc179735144)

[Single-Player Game 9](#_Toc179735145)

[Overview 9](#_Toc179735146)

[Story 9](#_Toc179735147)

[Hours of Gameplay 9](#_Toc179735148)

[Victory Conditions 9](#_Toc179735149)

[Artificial Intelligence 11](#_Toc179735150)

[AI Algorithms 11](#_Toc179735151)

[Game Engine 18](#_Toc179735152)

[The World Layout 19](#_Toc179735153)

[Game Characters 19](#_Toc179735154)

[20](#_Toc179735155)

[Reference List 22](#_Toc179735156)

Link To Git Repository - <https://github.com/Alissa442/games_development.git>

# Game Overview

## Common Questions

### What is the game?

**Starve Wars is an intense multiplayer survival game where players compete to be the last one standing in a rapidly depleting health environment. Each player begins with full health, which steadily declines, forcing them to scavenge for food to restore it. The objective is simple: collect food, avoid enemies, and outlast everyone else.**

**Players can find power-ups scattered across the map that grant temporary advantages, such as speed boosts, shields, or the ability to hinder opponents. Every decision matters as players must navigate the map strategically, avoiding enemies and environmental hazards while racing to secure critical resources.**

**As time passes, food becomes scarcer and the environment more hazardous, increasing the stakes and forcing players into closer, more dangerous confrontations. The last player to avoid starvation wins, making quick thinking and sharp reflexes essential for victory.**

**Additionally, players can unlock new levels as they progress, each with unique designs, challenges, and hazards, adding variety to the gameplay. Customization options, including character skins and power-up enhancements, provide personal touches and tactical depth to every match.**

### What is the target audience?

Starve Wars is designed for a broad range of players, primarily targeting mobile gamers between the ages of 12 and 35. This demographic includes both casual gamers looking for quick, fun sessions and competitive players seeking challenging gameplay. Our audience enjoys fast-paced, action-packed experiences that combine strategy, quick decision-making, and social interaction, making the game appealing to those who love multiplayer action, survival challenges, and competitive environments.

The game is designed to be accessible to both male and female players, catering to a diverse group of gamers who enjoy engaging with friends or competing against others online. Whether they’re playing solo or in a team, our audience values games that are easy to pick up but offer depth and challenge for long-term engagement. Starve Wars aims to provide an entertaining and immersive experience that fits into the busy, on-the-go lifestyle of mobile gamers.

### Why create this game?

We created **Starve Wars** out of our love for old-school retro games and the fun, casual experiences they provided. Those classic games had a charm that didn’t rely on overcomplicated mechanics but still managed to keep players engaged for hours. We wanted to bring that sense of fun back, but with a modern twist suited for today’s mobile gamers.

Additionally, we’re big fans of quick-play waiting games—those you can jump into during short breaks or while on the go. Our goal was to design a game that doesn’t take itself too seriously, offering a lighthearted atmosphere while still keeping players on their toes with strategic gameplay. **Starve Wars** combines the simplicity and charm of retro games with fast-paced action and competitive multiplayer elements, making it the perfect mix of casual fun and engaging challenge.

We saw an opportunity in the market for a game that’s easy to pick up but hard to put down—one that caters to both casual players and those seeking more intense competition. Whether you’re playing for a few minutes or diving into longer sessions with friends, **Starve Wars** offers an entertaining, approachable experience for all types of gamers.

### Where does the game take place?

Describe the world that your game takes place in. Simple as that. Help frame it in the reader’s mind by spending a few sentences on it here. You can go into lengthy detail later in a section solely dedicated to describing the world. Remember that we want to keep this part of the design light and readable.

### What do I control?

In Starve Wars, you control a single character—a small, feisty wrestler who is fighting for survival in a world full of enemies. Your wrestler must navigate the environment, collect food to replenish health, and dodge enemies trying to sabotage your efforts. As the game progresses, you’ll need to make quick decisions, using both strategy and reflexes to stay alive and outlast your opponents. The character is customizable, allowing players to personalize their wrestler with different skins and accessories.

### How many characters do I control?

You control just one character at a time in Starve Wars—the wrestler. The focus is on the individual player’s survival, so each player is solely responsible for their wrestler’s fate, relying on strategic movement and resource collection to stay in the game. The control scheme is designed to be simple and intuitive, giving players the tools they need to manage their wrestler’s actions, from collecting food to using power-ups, with responsive, fast-paced gameplay.

### What is the main focus?

In Starve Wars the primary focus is for players to progress through the various levels of the arena, all while trying to outlast their enemies. Each level presents new challenges, including scarce resources and increasing dangers. Players must carefully navigate the arena, collecting food to prevent their health from depleting and using power-ups to gain an edge over their opponents. The objective is not just to survive, but to be the last player standing by outmaneuvering and outsmarting both the environment and other players. As players advance, levels become more difficult, requiring sharper strategy and quicker reflexes to succeed. The ultimate goal is to conquer all the levels and be the last survivor in the arena.

### What’s different?

Starve Wars sets itself apart with its lighthearted and humorous approach to survival games. While many games in the market focus on intense combat or serious tones, Starve Wars blends quirky humor with fast-paced action. Players control small, cartoonish wrestlers fighting to stay alive, creating a fun and comical environment where even the struggle for survival has a playful edge.

The game’s whimsical art style and exaggerated animations keep things entertaining, and the addition of power-ups and fun character customizations adds to the humor. Instead of grim battles, players find themselves in over-the-top, chaotic arenas where the race for food and survival becomes a frantic yet enjoyable experience. This mix of fun visuals, easy-to-learn mechanics, and fast-paced strategy makes Starve Wars stand out as a refreshing, more casual take on the survival genre.

# Team Roles

Individual Member Roles:

Alissa Borg & Michael Puccini (Collaborative Designers, Writers, and Developers): Both Alissa and Michael worked closely together throughout the entire development of Starve Wars. They collaborated on designing the gameplay mechanics, writing the game design document, and coding the game’s core functionality.

The responsibilities were shared equally, with both members contributing to the writing, coding, and testing phases. They worked as a team to create the flowcharts, AI algorithms, and game structure, ensuring that every element of the game was well-rounded and cohesive. While testing was a joint effort, they frequently alternated roles—one focusing on writing and design while the other tested and refined the mechanics.

Together, they ensured that every aspect of the game, from the concept to the final testing, reflected their collective vision and teamwork.

# Feature Set

## General Features

* Health-Based Survival Mechanic: Players must continuously collect food to replenish their rapidly depleting health, creating a sense of urgency throughout the game.
* Power-Ups and Boosts: Collecting power-ups gives players temporary advantages, such as increased speed, invincibility, or hindering opponents.
* Dynamic Environments: Levels become progressively harder with more enemies and obstacles as players advance, adding complexity and keeping the game engaging.
* Progression System: Players can unlock new, more challenging levels as they progress, providing incentives to keep playing and improving.
* Simple, Intuitive Controls: The game is designed with easy-to-learn controls, ensuring that both casual and experienced gamers can pick up the game quickly.
* Fun Art Style and Humor: The game's light-hearted, cartoonish art style, paired with its quirky characters and setting, keeps the tone fun and engaging.

## Gameplay

**Instructions to Play**

In **Starve Wars**, your goal is to survive and outlast your enemies while completing specific objectives to win each level. The game is played in a large arena where you control a small dragon character and must collect power-ups, food, and other items, while avoiding enemies and hazards. Here's how to play:

**Basic Controls:**

* **Move**: Use the **WASD** keys or **Arrow Keys** to move your character around the arena.
* **Collect**: Approach food, power-ups, or items, and your character will automatically pick them up.
* **Deliver**: Collect food boxes and deliver them to the chef, usually located in one of the corners of the arena.

**How to Win:**

To win each level, you must complete the specific objectives given to you. These objectives vary by level but generally include:

1. **Collect Food**: Gather the required number of food items scattered around the arena.
2. **Collect Power-Ups**: Boost your stats by picking up power-ups that enhance speed, health, or other abilities.
3. **Deliver Food to the Chef**: After collecting enough food boxes, deliver them to the chef in the designated delivery area.
4. **Survive**: Outlast all enemies by avoiding or defeating them. Some levels may require you to survive until all enemies are eliminated.
5. **Speed**: In some levels, you’ll need to increase your speed to a certain level by collecting speed power-ups.
6. **Time Limit**: Complete all objectives within a specific time limit to win the level.
7. **Triggered Items**: Trigger a set number of mines or freeze items as part of the victory conditions.

Once all objectives for a level are completed, you will be declared the winner, and the level will end in victory.

**How to Lose:**

You can lose the game in the following ways:

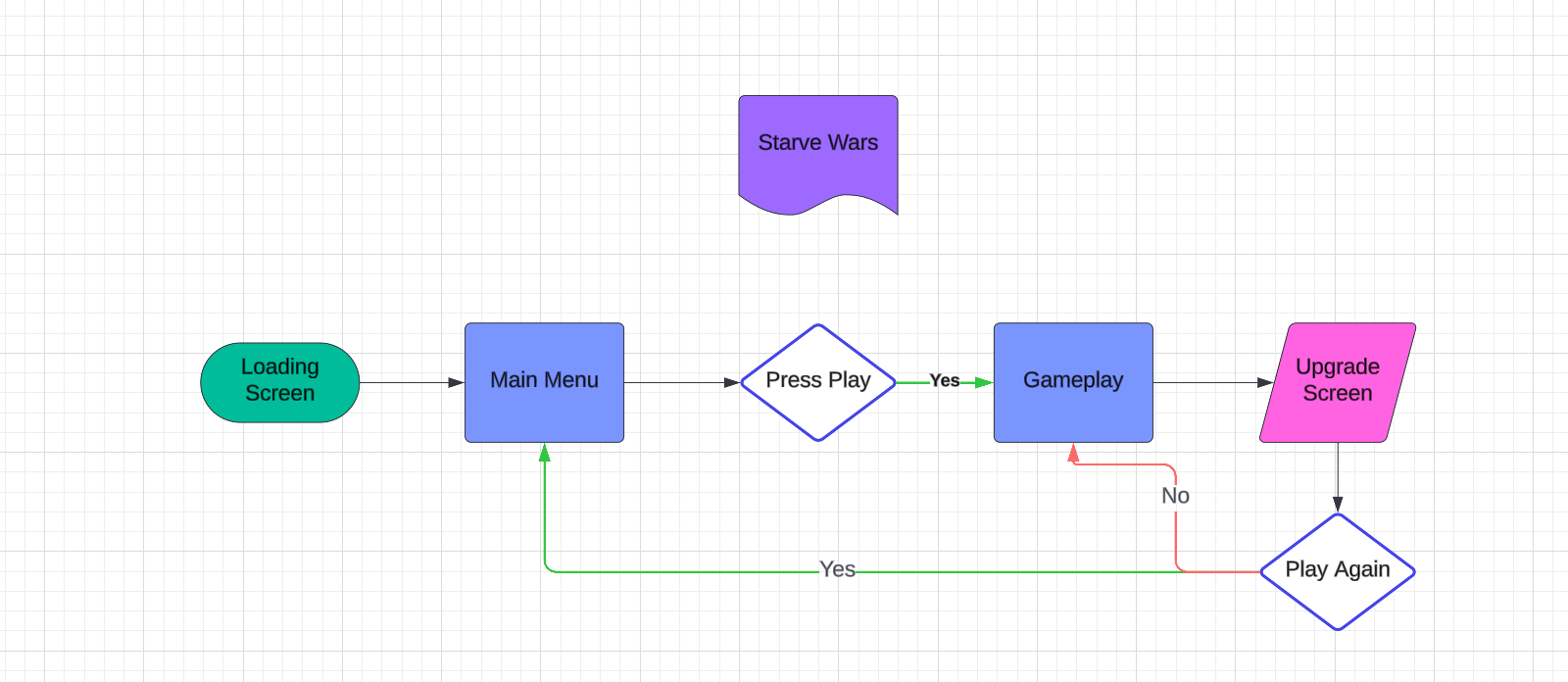
1. **Health Depletion**: If your health drops to zero (from enemy attacks, hazards, or lack of food), your character will die, and the level will end in failure.
2. **Failure to Meet Objectives**:
   * If you run out of time before completing the level’s objectives, the game will mark the level as failed.
   * Some objectives, like staying alive (DoNotDie condition), must be maintained throughout the level. If you die or fail to meet these key objectives, you will lose.
3. **Not Collecting Required Items**: If you fail to collect the required number of items (such as food, gems, or ingredients), you won’t be able to win the level, resulting in failure.

**Tips for Success:**

* **Stay Mobile**: Constantly move around the arena to avoid enemies and gather items quickly.
* **Collect Power-Ups**: Make sure to gather power-ups to boost your speed, health, and other abilities.
* **Watch the Timer**: Keep an eye on the time limit, and prioritize completing the most important objectives first.
* **Outlast Enemies**: Some levels require you to outlast all your opponents. Stay alive and let enemies fight each other if necessary.

By managing your resources, completing objectives, and staying alive, you can win each level and progress in **Starve Wars**!

## Flowchart

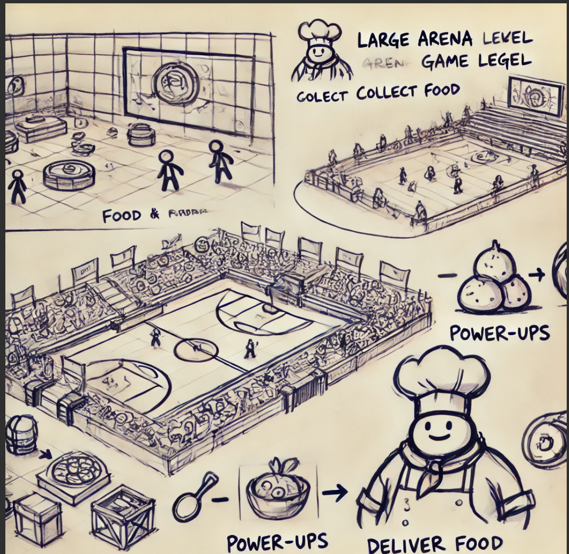


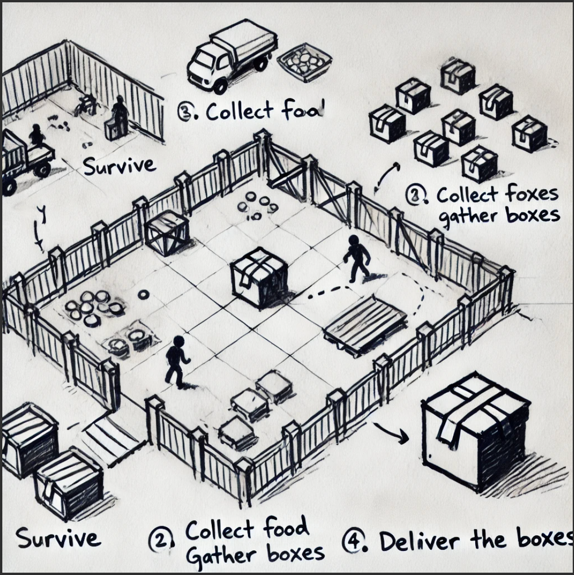
## Storyboards

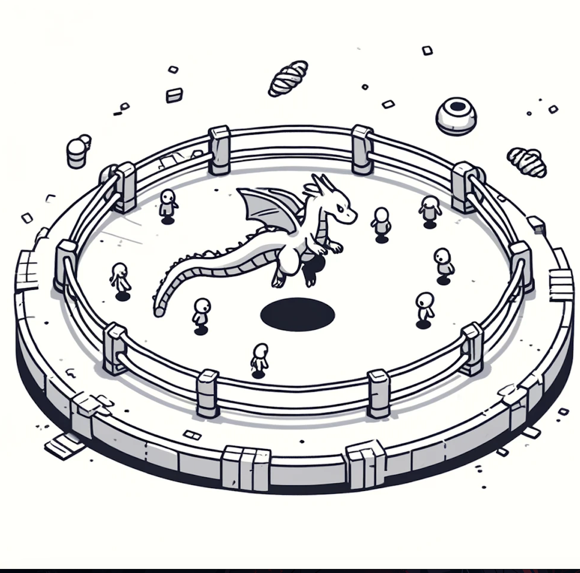
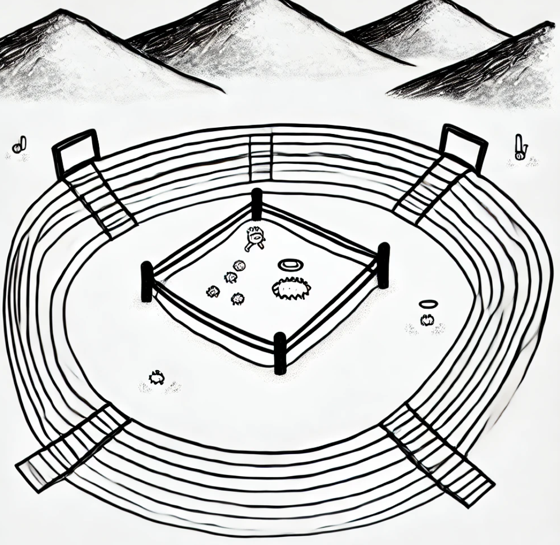
A screenshot of a video game

Description automatically generated

Early level designs

Design 1 -





## 

## Controls

Player Controls:

1. Movement:

- The player can move using the WASD or Arrow Keys.

- The movement is based on the Horizontal (left-right) and Vertical (forward-backward) inputs.

- The player’s movement speed is controlled by the `Speed` component.

- Smooth Turning: When the player moves, their character smoothly rotates to face the direction of movement using smooth transitions.

2. Manual Control State:

- In ManualControl state, the player has direct control over the character’s movement.

- The movement is handled in a top-down 2D style, meaning movement is along the X and Z axes.

- The speed at which the player moves is determined by the `Speed` component, which can be influenced by various upgrades or abilities.

3. Speed and Research Effects:

- Speed and other abilities are enhanced by player research upgrades. For example:

- FasterShoes: Increases the player's movement speed based on the level of research.

- DietPills: Further increases speed but decreases health as a trade-off.

- RubberArms: Increases the player’s range.

- BetterDigestion and GutHealth: Affect health recovery and poison resistance.

4. Health Management:

- The player’s health is managed by the Health component, which integrates with various upgrades.

- If the player dies (health reaches zero), a death event is triggered, and the player's death is handled by invoking specific game global events.

5. Player Abilities:

- Additional upgrades affect the player’s abilities (e.g., healing, poison resistance, or power-up durations).

- The upgrades and abilities are applied using PlayerPrefs, allowing the player to unlock and improve abilities over time.

Additional Control Details:

- The player can’t move if the Speed component’s `canMove` flag is set to `false` which occurs when player is stunned.

- Rotation and movement are smoothly interpolated to provide a responsive feel for the player’s character as it navigates the arena.

# Single-Player Game

## Overview

In **Starve Wars**, the single-player experience is a fast-paced survival challenge where players must navigate through increasingly difficult levels, collecting food to replenish health and avoiding enemies. The goal is to outlast the environment and progress through each level while managing resources carefully. The game encourages strategic movement and quick decision-making to stay alive in a dynamic, ever-changing arena.

**Single Player Game Detail #1**

Level Progression: The single-player game features a series of levels, each becoming progressively harder. As players move forward, enemies become more aggressive, and food becomes scarcer. This keeps the player constantly challenged and forces them to adapt to new obstacles and threats. Power-ups and hidden items are scattered throughout the levels, providing temporary advantages and encouraging exploration.

**Single Player Game Detail #2**

Survival and Resource Management: The core mechanic revolves around managing the player’s health by collecting food while avoiding enemies. Players must balance the urgency of replenishing health with the need to avoid danger, creating a tense and rewarding experience. Additional resources like power-ups or rare items can provide critical boosts at key moments.

## Story

**Starve Wars** follows a humorous, light-hearted narrative where players are thrust into a competitive survival arena where only the strongest can survive. You are a small wrestler battling not only the environment but also other creatures in the quest for food. Along the way, you'll encounter quirky characters and face obstacles designed to test your strategic thinking and reflexes. The story focuses more on survival and competition than deep narrative, keeping the tone playful. More story details can be found in the appendix or a separate document, especially if expansions or story arcs are added later.

## Hours of Gameplay

The main single-player experience is designed to be short, with individual levels taking only a few minutes to complete. However, each level offers a lot of replayability, allowing players to come back and improve their performance, beat personal bests, and uncover new strategies. The overall game can provide 5-8 hours of playtime, depending on how much the player explores and replays levels for higher mastery.

## Victory Conditions

In the game, each level has unique victory conditions that the player must meet to win. These conditions are handled by different scripts that track the player's progress toward specific objectives, such as collecting food, gems, ingredients, or poison.

Each of these conditions is tied to a UI component that updates the player's progress, showing how much of each objective they’ve completed and whether they’ve won or lost based on their performance.

**1. CollectXFood:**

* **Objective**: The player must collect a certain number of food items to win.
* **Mechanics**: Once the required number of food items is collected, the game considers this condition met, and the victory UI is updated.
* **UI Displays**: Eat Food (Collected / Total).

**2. CollectXGems:**

* **Objective**: The player must collect a specific number of gems.
* **Mechanics**: After collecting enough gems, the condition is fulfilled, updating the UI accordingly.
* **UI Displays**: Collect Gems (Collected / Total).

**3. CollectXIngredients:**

* **Objective**: The player must collect a certain number of ingredients, typically to progress through the level or prepare something for the chef.
* **Mechanics**: The condition is completed when the required number of ingredients is gathered.
* **UI Displays**: Get Boxes (Collected / Total).

**4. CollectXPoison:**

* **Objective**: The player must collect a certain number of poison items.
* **Mechanics**: The game tracks the number of poison items collected, and the condition is marked as complete when the required amount is reached.
* **UI Displays**: Eat Poison (Collected / Total).

**5. DoNotDie:**

* **Objective**: The player must stay alive throughout the level.
* **Mechanics**: The player’s health is monitored, and if they die, the condition is marked as failed, and the game ends. The HasLost state is triggered upon death.
* **UI Displays**: None directly, but the player’s life status is central to this condition.

**6. GetXSpeed:**

* **Objective**: The player must increase their speed to a specific target.
* **Mechanics**: The player's speed is tracked, and the condition is fulfilled once the player's speed reaches or exceeds the target.
* **UI Displays**: Get Speed (Current Speed / Target Speed).

**7. OutliveAllYourOpponents:**

* **Objective**: The player must survive longer than all enemies in the level.
* **Mechanics**: The game tracks the number of enemies and the player. If the player outlasts all enemies, this condition is met. If the player dies, the condition is failed.
* **UI Displays**: None, but enemy count can be used to track progress.

**8. Rewards:**

* **Objective**: The player can collect rewards based on their performance, which may be influenced by research upgrades.
* **Mechanics**: The reward amount is displayed and updated based on the player's progress and any research upgrades that increase reward levels. This is more of a motivational factor than a direct win condition.
* **UI Displays**: Rewards (xAmount).

**9. TimeLimit:**

* **Objective**: The player must complete all tasks within a specific time limit.
* **Mechanics**: A countdown timer is displayed, and if time runs out before all tasks are completed, the condition fails.
* **UI Displays**: Time Limit (Time Remaining / Time Limit).

**10. TriggeredXFreeze:**

* **Objective**: The player must trigger a specific number of freeze items.
* **Mechanics**: The game tracks how many freeze consumables the player uses, and the condition is fulfilled when the required number of uses is reached.
* **UI Displays**: Use Freeze (Triggered / Required).

**11. TriggeredXMines:**

* **Objective**: The player must trigger or set off a specific number of mines.
* **Mechanics**: The game tracks the number of mines triggered by the player, and the condition is fulfilled when the required amount is reached.
* **UI Displays**: Set Off Mines (Triggered / Required).

How Victory Conditions Work:

- Each WinCondition tracks the progress of a specific objective.

- The player interacts with items in the game world (e.g., food, gems, poison), and upon collection, the relevant WinCondition updates the UI with the current progress.

- Listeners\*are used to detect when a player picks up specific items (e.g., food, gems, etc.). The player’s progress is updated in real time, and when the required number of items is collected, the victory condition is met.

- Once all win conditions for a level are satisfied, the player is declared the winner for that level.

Ui Feedback:

- The game's UI provides immediate feedback to the player on their progress toward each objective. Each type of item (food, gems, etc.) has its own counter displayed on the screen, which is updated as the player progresses.

- If the player meets all the conditions required for that level, a victory message is displayed.

# Artificial Intelligence

## AI Algorithms

**ISTATE Machine -** The IStateMachine interface defines the foundational structure for state machines within the game’s AI system. It establishes a standard set of methods that state machines must implement, ensuring consistency in how different AI characters transition between states, interact with targets, and utilize navigation.

1. SetState(IState newState)` – This method allows the AI entity to switch to a new state. By providing the `newState`, the AI can change its behavior dynamically, transitioning from idle to chasing, attacking, or other predefined actions.

2. `GetState()` – This method returns the current state of the AI. It is essential for tracking what behavior the AI is currently performing, which can inform other decisions or trigger additional transitions.

3. GetAgent()– This method retrieves the NavMeshAgentcomponent of the AI character. The NavMeshAgentis responsible for handling pathfinding and movement, allowing the AI to navigate towards targets or move through complex environments.

4. `GetTarget()` – This method returns the current target of the AI. The target can be an object, an enemy, or even food. The AI relies on this to determine what it should be interacting with or moving towards.

5. `SetTarget(GameObject newTarget)` – This method sets a new target for the AI, updating what it will focus on or move towards. For example, when an enemy or a food item appears, the AI can switch its attention to the new target.

This interface ensures that all state machines have a common framework for managing transitions and behavior, making the AI system modular and flexible. The use of NavMeshAgent in conjunction with target management allows for sophisticated navigation and interaction, helping various AI entities efficiently respond to changing game conditions.

**State Machine** -The StateMachine class serves as the foundation for various AI entities in the game, enabling them to transition between different behaviors and manage their navigation. It’s an abstract class that provides the core functionality required for AI characters to switch between states, move towards targets, and respond to changes in the game environment.

Key components of the StateMachine:

1. State Management – The AI operates on a state-based architecture, where each state defines a specific behavior for the AI entity (e.g., searching for food, attacking, idle). The currentState tracks the active behavior, while defaultState represents the AI’s default or fallback state. When the AI needs to switch behaviors, the ChangeState method is used to update the currentState to a new one, and the AI's currentStateName is updated accordingly.

2. NavMeshAgent Integration – The NavMeshAgent is responsible for pathfinding and movement within the game world. By setting destinations through the agent, the AI can move towards specific locations or targets, such as food or enemies. If the currentState is null (e.g., no target or action), the AI stops moving by setting its destination to its current position.

3. Event-Driven Behavior – The StateMachine listens for global game events, such as when a consumable is picked up. Through the onConsumablePickedUpByWhom event, the AI can track who picked up the last consumable. This functionality is key to certain AI behaviors, such as determining if an entity should react based on whether a player or another AI picked up the consumable. The AI’s health is also updated dynamically using an event listener tied to the \*\*Health\*\* component, allowing the AI to react based on its health state (e.g., entering a retreat or healing state).

4. State Transition and Behavior Execution – In the Update method, the AI constantly checks if a state is active. If no state is active, the AI will stop its movement and revert to the defaultState. The Tick method of the currentState is called each frame, executing the logic associated with that state (e.g., moving, attacking, or searching).

5. Health Management – The AI tracks its health through the health variable, which is updated via the onHealthChange event listener. This allows the AI to transition between states based on its health (e.g., switching to a defensive or aggressive state when its health changes).

The StateMachine is the backbone of AI behavior, providing the framework for managing states, movement, and event-driven interactions. It allows for flexible, modular AI, where different behaviors can be plugged into the state machine, making it easy to extend and customize the AI for different characters and situations.

**Chef State Machine -**The **ChefStateMachine** AI operates on a **state machine algorithm**. State machines are a commonly used AI architecture in games, where an entity switches between different states based on conditions or inputs. In this case, the Chef alternates between three key states:

1. **ChefIsWaiting** – The default state where the Chef is idle, waiting for ingredients to be supplied by other entities.
2. **Cooking** – Once ingredients are received from suppliers (other enemies or players), the Chef switches to the Cooking state. In this state, it processes the ingredients over a specified period (3 seconds) to produce food items.
3. **ChefIsRaging** – If the Chef doesn’t receive enough ingredients within a certain timeframe (120 seconds), it becomes enraged. In this enraged state, the Chef starts dealing continuous damage (5 damage per second) and requires more aggressive handling from players.

The transitions between these states are determined by the conditions, such as the availability of ingredients or the elapsed time before the Chef enters the enraged state.

**Queue Mechanism:** The Chef uses a queue system to manage the suppliers (enemies or players) that provide ingredients. Each time an ingredient is delivered, the supplier is added to the queue, and the ingredient count is increased.

**Trigger System:** When another object carrying ingredients enters the Chef’s trigger collider, the Chef checks whether the object can supply ingredients (through the IngredientsCarrier component). If so, the Chef adds them to its queue and processes the ingredients for cooking.

This AI ensures that the Chef responds dynamically to the game’s environment, either cooking food from available resources or entering a dangerous rage if neglected for too long.

**Food State Machine -** The FoodieStateMachine employs a simple state machine algorithm designed to simulate a food-seeking behavior. The AI transitions between two main states, mimicking a basic search-and-chase dynamic, where the enemy is always driven by the need to find and move towards food. The states are as follows:

1. InefficientSearchForFood – This is the default state of the Foodie, where it wanders or searches for food in an inefficient, random manner. If there is no food detected, the AI will stay idle, staring at a wall or performing minimal movements until food appears in its vicinity.

2. MoveTowardsTarget – Once food is detected, the Foodie transitions to the MoveTowardsTarget state. In this state, it uses the Unity NavMeshAgent to move towards the food. The agent calculates a path and ensures the Foodie reaches its target.

The state transitions are determined by whether food is present or not:

- If the Foodie detects food while in the \*\*InefficientSearchForFood\*\* state, it transitions to MoveTowardsTargetto chase the food.

- If there is no food, the AI transitions back to InefficientSearchForFood, where it either wanders or waits until food becomes available.

The behaviour is intentionally simple but reflects a core survival instinct of chasing food when it's available and remaining idle when it’s not. This minimalist design makes the AI predictable but still fun, as players can manipulate its behavior by controlling the availability of food in the environment.

**Mine Layer State Machine** – The MineLayerStateMachine uses a state machine algorithm that governs the behavior of an AI enemy designed to lay mines in strategic locations while managing its own survival. This state machine controls how the enemy alternates between laying mines, searching for food, and moving towards targets. The AI can make decisions based on its health and the need to plant traps (mines) for enemies or other players.

The key states and transitions are:

1. LayAMine – This is the default and primary state where the enemy plants mines in the environment. The enemy uses a pool of mine prefabs (predefined mine objects) and lays a mine after a set time delay (3 seconds). The AI enters this state as long as it has enough health to maintain the activity. Mines are laid in strategic locations to hinder the player’s progress or set traps.

2. InefficientSearchForFood – If the enemy's health drops below a certain threshold (15 or lower), it will stop laying mines and transition to this state. In this state, the enemy searches for food in an inefficient and random manner, attempting to replenish its health.

3. MoveTowardsTarget – Once food is detected during the inefficient search, the AI transitions to this state, where it uses a NavMeshAgent to move towards the food and collect it. After reaching the food and replenishing its health, the AI returns to the LayAMine state to resume laying traps.

State Transitions:

- LayAMine → InefficientSearchForFood: This transition happens when the enemy’s health drops below a set threshold (15).

- InefficientSearchForFood → MoveTowardsTarget: This transition occurs when the AI detects food during the search.

- MoveTowardsTarget → LayAMine: Once food is collected, the AI returns to laying mines.

Health-Based Decision Making: The AI uses health as a key factor to determine when to stop laying mines and search for food, introducing an element of resource management. The enemy dynamically switches between defensive (mine-laying) and survival (food-searching) behaviors.

This approach creates a challenging and adaptive enemy AI that both lays traps and seeks to sustain itself, making the game environment more unpredictable and dangerous for players.

**Sous Chef State Machine** -The SousChefStateMachine AI plays a supportive role, primarily assisting the Chef in the game. It operates within a relatively simple framework, managing the delivery of ingredients and food to specific locations while also reacting to the state of the Chef.

Key elements of the AI’s behavior:

1. Delivery System – The Sous Chef AI is responsible for delivering food and ingredients to a designated location, represented by the deliveryLocation. It works in tandem with the Chef by ensuring the necessary ingredients are available for the Chef to continue cooking. This introduces a cooperative dynamic between AI characters.

2. Chef State Awareness – The Sous Chef constantly checks the state of the ChefStateMachine to determine if the Chef is enraged. If the isChefRaging flag is set to true, the Sous Chef may alter its behavior to prioritize specific tasks, such as gathering ingredients faster or focusing on calming the enraged Chef.

3. Spawner Interaction – The Sous Chef AI interacts with two Spawner objects:

- foodSpawner: Spawns food items, which the Sous Chef can deliver to the Chef.

- poisonSpawner: Spawns poisoned items, which may require the Sous Chef to manage or avoid delivering to the Chef, introducing another layer of decision-making.

The SousChefStateMachine does not currently include explicit states or transitions, but it works as a utility AI in service to the Chef, responding to the state of the Chef and fulfilling the delivery of necessary resources. The AI can easily be extended to introduce more complex behavior, such as deciding between delivering food or poison or prioritizing actions based on the Chef’s health or rage level.

**Wing Baby State Machine** -The WingeBabyStateMachine is built upon a state machine algorithm that controls the behavior of an AI character called "Winge Baby." This AI alternates between searching for consumables and reacting with tantrums based on specific thresholds, making it an unpredictable and dynamic entity in the game.

Key components of the AI's behavior:

1. LookForConsumableBalanced – This is the default state in which the Winge Baby searches for consumables within the environment. It’s a balanced state, indicating that the AI is not yet frustrated but is actively looking for food or other necessary items.

2. WingeMoveTowardsTarget – When a consumable is detected, the AI transitions into this state. The NavMeshAgent directs the Winge Baby towards the consumable, and the AI attempts to collect it. This state incorporates thresholds for triggering the AI’s emotional reactions.

- Nerd Rage Threshold (10): If the AI’s frustration reaches this threshold, it may transition into a more aggressive or desperate state.

- Tantrum Threshold (3): If the AI’s frustration level drops below this threshold (e.g., after failing to collect consumables repeatedly), it will transition to the Tantrum state.

3. Tantrum – When the AI becomes too frustrated (by not finding or obtaining consumables), it enters the Tantrum state. In this state, the Winge Baby behaves erratically, likely throwing a tantrum that lasts for a set duration (3 seconds in this case). During this state, the AI might behave aggressively, creating chaotic moments in the game. Once the tantrum ends, the AI transitions back to the \*\*LookForConsumableBalanced\*\* state to resume its search for food.

State Transitions:

- LookForConsumableBalanced → WingeMoveTowardsTarget: When a consumable is found, the AI shifts to this state to move toward the target.

- WingeMoveTowardsTarget → Tantrum: If frustration builds or the AI fails to meet its thresholds, it enters the Tantrum state.

- Tantrum → LookForConsumableBalanced: After the tantrum ends, the AI returns to its balanced state to resume searching for consumables.

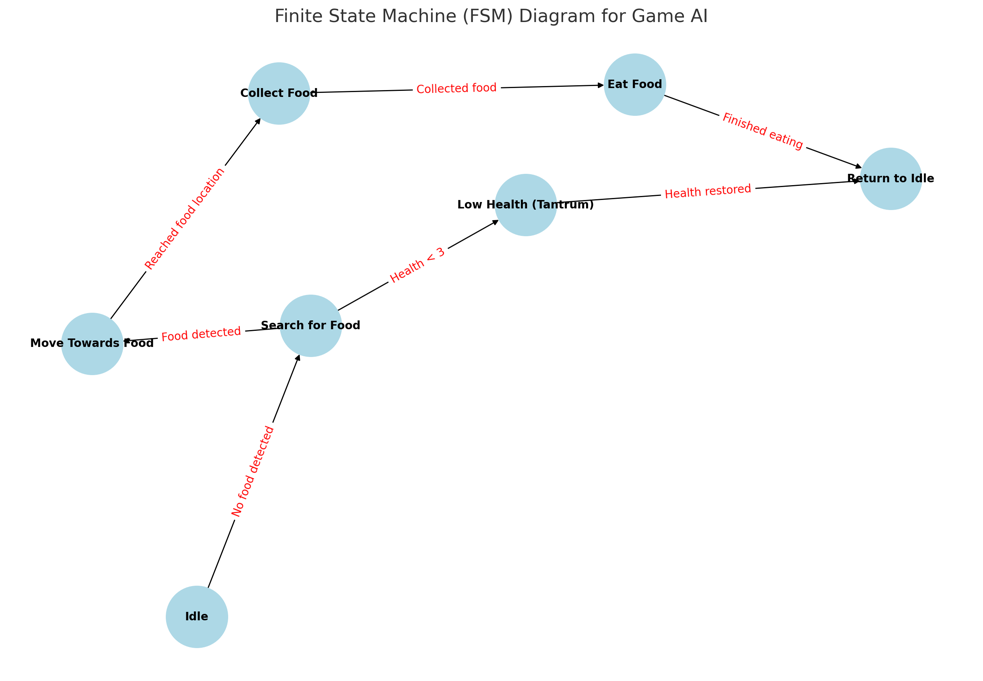
Threshold-Based Behavior: The AI has two emotional thresholds:

- Nerd Rage Threshold (10): When the AI’s frustration or need grows too high, it reacts more aggressively or desperately.

- Tantrum Threshold (3): Below this threshold, the AI throws a tantrum in frustration, creating more chaotic behavior.

This combination of emotional thresholds and state transitions makes the \*\*WingeBabyStateMachine\*\* highly reactive to its environment and resources, simulating an emotional response to failure or success in finding consumables.

**FSM / Ai Map**

****

**The Game World**

**Overview**

The game world of Starve Wars is a dynamic, fast-paced environment designed to challenge players’ survival instincts. It is set in an arena where players must compete to outlast enemies by collecting food and avoiding hazards. The world is designed to be engaging, filled with power-ups, food boxes, and enemies that create both opportunities and threats. Each arena is crafted with strategic elements to keep players on their toes as they navigate and compete in intense matches.

**Arena Battlefields**

Each arena in Starve Wars is designed as a confined space with multiple obstacles, food items, and power-ups. The arenas are enclosed by walls or barriers, ensuring players are always near enemies, adding to the game's intensity. The key feature of the arena is its tactical layout, which encourages strategic movement and resource management.

**Dynamic Gameplay Elements**

The game world is filled with various dynamic elements such as:

* Food Drops: Scattered across the arena, food keeps players alive as they collect it to maintain their health.
* Power-ups: These give players temporary boosts like increased speed, attack power, or health regeneration.
* Enemy AI: The world is populated with aggressive enemies that patrol the arena, posing a constant threat.

**The Physical World**

Overview

The physical world of Starve Wars is centered around a series of themed arenas. These arenas are built to reflect the different challenges and dynamics that players will face, from open spaces to tight corners where enemies lurk. The layout of each arena encourages fast-paced, strategic gameplay where positioning and movement are key to survival.

**Key Locations**

* The Main Arena: The primary battle location where players collect food and power-ups while avoiding enemies. This is where the majority of gameplay takes place.
* Chef's Corner: A designated area in the arena where players can deliver food boxes to gain points or bonuses. It's a safe zone but also a strategic point where players may be ambushed.
* Enemy Spawn Zones: Marked areas where enemies enter the arena. Players need to avoid these areas to prevent being overwhelmed.

**Travel**

* Player Movement: Players move around the arena using basic controls, such as the WASD or arrow keys, to navigate in a top-down view. Speed can be boosted through power-ups, allowing players to outrun enemies or reach food quicker.
* Enemy Movement: Enemies move on pre-determined paths or chase the player once detected, forcing players to use strategy and timing to avoid them.

**Scale**

The arena size is designed to keep players and enemies in close quarters, with a moderate scale that promotes frequent encounters. The world’s scale is perfect for quick, engaging matches where there’s little room to hide, encouraging constant movement and quick decisions.

**Objects**

* Food Items: These are scattered around the arena and must be collected to keep the player's health from dropping.
* Power-ups: Temporary boosts that help players in combat or enhance movement speed.
* Crates: Some food comes in crates that must be delivered to the chef, adding a layer of strategy as players juggle gathering food and delivering it safely.
* Hazards: There are environmental traps or objects that can damage the player if they aren’t careful.
* For a full list of objects, refer to the Objects Appendix.

**Time**

Time in Starve Wars is critical. Each match has a time limit, and players must collect food and power-ups or defeat enemies before time runs out. Time is a central element in determining match strategies and victory conditions.

**Camera**

**Overview**

In Starve Wars, the camera is positioned at an 80-degree angle to provide an isometric view of the arena. This perspective allows players to see both the immediate area around their character and the battlefield at large. The camera's setup ensures that players can navigate the arena efficiently while maintaining situational awareness of enemies, food, and power-ups.

**Isometric Angle Camera**

Positioning: The camera is angled at 80 degrees above the arena, offering a near top-down but slightly angled view. This provides a mix of both vertical and horizontal perspectives, allowing players to see the depth of the arena as well as the layout of obstacles and enemies.

Static Camera: The camera does not follow the player's movements; it remains in a fixed position over the arena. This ensures players always have a complete view of the entire gameplay area without the need for camera controls or adjustments.

## Game Engine

**Overview**

Starve Wars is built using the Unity Engine, a powerful and widely adopted platform for developing 2D and 3D games. Unity offers a comprehensive set of tools for real-time rendering, physics simulations, and cross-platform game development, making it an ideal choice for this project. Its component-based architecture simplifies the management of objects, characters, and their interactions within the game world.

**Game Engine Management**

The Unity engine is designed to manage all aspects of the game world. This includes monitoring the state of each object, character interactions, and environmental dynamics. By using Unity’s efficient update systems, we ensure that game elements respond in real-time to player actions and world events, creating a seamless and engaging experience.

**Collision Detection**

In Starve Wars, we implemented collision detection using Unity's built-in physics engine. Unity's physics system ensures that objects interact realistically and accurately. The key components we used for collision detection include:

Collider Components: We added Collider components to objects to define their physical boundaries. By configuring the "Is Trigger" option, we allowed some objects to act as trigger zones rather than solid barriers, ideal for detecting player interactions without obstructing movement.

Rigidbody Components: Objects involved in collisions have Rigidbody components, allowing the physics engine to calculate forces, movement, and interactions between objects accurately.

OnTriggerEnter Method: This method was utilized to detect when an object entered a trigger zone. It allowed us to implement features like item pickups, interaction zones, and collision-based events, giving the game dynamic behavior based on player actions.

# The World Layout

The game is set in a sprawling football stadium situated in the heart of a barren desert, a stark symbol of a once-thriving civilization now overshadowed by desolation. The vast expanse of endless sand stretches beyond the stadium, creating an unbroken floor of golden grains that seem to swallow everything in their path. This desert backdrop adds to the sense of hopeless isolation, with only a solitary road passing by a rundown gas station—a road that leads nowhere, enhancing the eerie feel of a forgotten dystopian event town.

Word Detail 1: Endless Sand Floor

The desert surrounding the stadium is characterized by a vast, unrelenting sea of sand. The floor of the landscape is a golden, shimmering expanse that seems to stretch out endlessly in every direction. The sand gives a suffocating sense of isolation and emptiness, as if the entire world has been consumed by this arid wasteland, with no signs of life beyond the stadium’s boundaries.

Word Detail 2: The Big Stadium and Warm Lighting

The stadium itself stands as a towering monument amid the emptiness, its colossal structure dominating the landscape. The warm glow of stadium lights bathes the field in a nostalgic, almost comforting light. However, this light contrasts sharply with the cold emptiness of the surrounding desert, as the massive stadium looms over an arena that once echoed with life but now feels abandoned, amplifying the game’s dystopian atmosphere.

# Game Characters

Overview

In Starve Wars, players take control of a small dragon that must outlast various quirky enemies in a fast-paced arena. The dragon is quick, agile, and equipped with abilities like speed boosts and item collection. Throughout the game, the dragon faces different enemies, each with unique behaviors and strategies designed to challenge the player's survival skills.

Creating a Character

The player's character is always the dragon, and while there are no unlockable skins or customization options, the dragon’s abilities and appearance are crafted to be fun and engaging right from the start. Players focus on using the dragon’s natural agility and speed to outmaneuver enemies and complete objectives.

Enemies and Monsters

1. The Chef (Scarecrow):

- A formidable Scarecrow Chef who roams the arena, setting traps and trying to stop the player from delivering food. He wields kitchen tools and is relentless in chasing down the player. His role is to be a constant threat, forcing players to strategize around his movements.

2. The Sous Chef (Clown):

- The Clown Sous Chef is a mischievous character with a knack for creating chaos, but behind the scenes, he operates with a priority queue for hunting resources. The Sous Chef prioritizes specific items based on their distance and a priority multiplier assigned to each type.



3. The Foodie (Angry Puppy)

- This Foodie Monster is a competitor, constantly searching for and eating food in the arena. The player must race against this enemy to grab food, as the Foodie can deplete vital resources that the player needs to survive.

4. The Minelayer (Bomb):

- A tactical bomb who patrols the arena and plants mines in strategic spots. Players must navigate carefully to avoid triggering these mines, or find ways to disable them. His role is to create hazards that can block the player’s path. Also have the poison layer which is a variant.

## 

5. WingeBaby (Crying Baby Monster):

- The WingeBaby is a dangerous foe whose power grows the more it cries. As the baby throws tantrums, it becomes more aggressive and difficult to deal with. Players must either defeat it quickly or avoid it as its attacks escalate over time.

Team Logs

|  |  |  |  |
| --- | --- | --- | --- |
| **October 13, 2024** |  |  |  |
|  |  |  |  |
| **Task Description** | **Due** | **Status** | **Notes** |
| Altered animations for enemies, chef, and player | October 13, 2024 | Completed | Animations now work as intended. |
| Clamped player movement to the play area | October 13, 2024 | Completed | Ensured player stays within boundaries. |
| Added Level 8 and visual fixes for Levels 5 and 6 | October 13, 2024 | Completed | Enhanced visuals for a better experience. |
| Fixed multiple pickup bug for consumables | October 13, 2024 | Completed | Improved consumable pickup system. |
| Main menu scrolling and icon updates | October 13, 2024 | Completed | Updated to a scrolling format. |
| Loading screen and animations added | October 13, 2024 | Completed | Includes a skip feature. |
| Updated GDD Template | October 13, 2024 | Completed | Added refinements and additional content. |
| Implemented Sous Chef state machine and Level 5 integration | October 13, 2024 | Completed | Sous Chef AI implemented. |
|  |  |  |  |
| **October 12, 2024** |  |  |  |
|  |  |  |  |
| **Task Description** | **Due** | **Status** | **Notes** |
| Added background to Main Menu and Upgrades Menu | October 12, 2024 | Completed | Improved UI design. |
| Completed Levels 1-4 | October 12, 2024 | Completed | First set of levels created. |
| Implemented character animations (WIP) | October 12, 2024 | In Progress | Animations need further refinement. |
| Built tiny town and stadium in the desert | October 12, 2024 | Completed | Environment design for dystopian setting. |
| Added arena assets and textures | October 12, 2024 | Completed | Textures and assets added to the barriers. |
| Added Freeze, Speed Boost, and Poison models/effects | October 12, 2024 | Completed | Models integrated into gameplay. |
|  |  |  |  |
| **October 11, 2024** |  |  |  |
| **Task Description** | **Due** | **Status** | **Notes** |
| Implemented WinConditions for Do Not Die, CollectXGems | October 11, 2024 | Completed | Added new win conditions for levels. |
| Developed Chef AI state machine | October 11, 2024 | Completed | Chef AI now functional. |

# Reference List

Unity Store:

Awesome Game UI pack 14  
<https://assetstore.unity.com/packages/2d/gui/icons/awesome-game-ui-pack-14-199366>

Bakery Object  
<https://assetstore.unity.com/packages/3d/props/food/3d-props-bakery-object-17167>

Worn Wooden Crate  
<https://assetstore.unity.com/packages/3d/props/furniture/worn-wooden-crate-246735>

3D Monster Bomb!!  
<https://assetstore.unity.com/packages/3d/characters/3d-monster-bomb-145319>

Action RPG FX  
<https://assetstore.unity.com/packages/vfx/particles/action-rpg-fx-38222>

Stone Floor Texture Tile  
<https://assetstore.unity.com/packages/2d/textures-materials/roads/stone-floor-texture-tile-18683>

Low Poly Barriers Pack Free  
<https://assetstore.unity.com/packages/3d/props/exterior/low-poly-barriers-pack-free-201810>

Restaurant Interior Full Pack  
<https://assetstore.unity.com/packages/3d/props/interior/restaurant-interior-full-pack-153273>