Name Carson Sloan Mark /50

## Brief introduction \_\_/3

I am in charge of initializing the game via a game manager script that provides accessors for necessary game settings, events and statistics. The game manager will control the flow of the game and call specific event scripts during each state of gameplay.

I will also be designing and building the map, which will have several unlockable areas and interactable objects (lab equipment, AC system, weapon shop), controlled by a map manager script.

## Use case diagram with scenario \_\_14

### Use Case Diagrams

Shape

Description automatically generated with medium confidence

A picture containing shape

Description automatically generated

### Scenarios

**Name:** **1** Player collides with dog

**Summary:** The player manager determines what should be done to the player when a collision is detected with a zombie dog.

**Actors:** Player

**Preconditions:** Game has begun, dogs have been spawned.

**Basic sequence:**

**Step 1:** Get the dog object that caused the event and get its attack value.

**Step 2:** Check the player’s status effects or armor values.

**Step 3:** Get the current difficulty (calculated by time survived and current area) from the GameManager object.

**Step 4:** Calculate how much health should be removed from the player as a result of this collision.

**Exceptions:**

**Step 4:** If the player is of type BCPlayer, then 0 health should be removed.

**Post conditions:** Player is invulnerable to zombie dog damage for 0.3 seconds, health bar updates.

**Priority:** 2

**ID:** D0

**Name:** **2** Get spawn locations for new zombie wave

**Summary:** When it’s time for a new wave of zombies to spawn, the locations to spawn them from must be calculated/chosen so they are spread out, but near the player and within accessible areas.

**Actors:** SpawnWave method

**Preconditions:** Game has begun, the timer has reached the next wave start time

**Basic sequence:**

**Step 1:** Check which areas are unlocked, and use that to add to the array of possible locations if needed.

**Step 2:** Iterate once through the spawn position array (looping back to the start if the end of the array is passed), and return that Vector2 position.

**Step 3:** Get the current player object from the GameManager (which is dynamic binding, as this object may be of type SurvivalPlayer or BCPlayer at runtime), and check its location.

**Step 4:** If the chosen spawn location is too far from the player, repeat step 2.

**Post conditions:** The next zombie in the wave will be spawned from this position.

**Priority:** 3

**ID:** Z1

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

### Diagram Description automatically generatedData Flow Diagrams

Data Flow Diagram 1, expanding on **Process 1: Instantiate Game**:

Diagram

Description automatically generated

### Process Description

**Process 1.4.1: Spawn Enemy Wave**:

1. Check if it’s time to spawn a new normal wave, using the integer number of seconds the game has been running.
   1. Check if (time > 200) { if (time % 5 == 0) { spawnWave(); }}
   2. else if (time % (12 - 2 \* (time / (30 + (time / 20))) == 0) { spawnWave(); }
   3. This means the time between waves will increase slowly, from every 12 seconds at first, to a minimum of every 5 seconds at 200 seconds.
2. If it’s time, check the getNextSpawnLocation() function for the Vector2 coordinate where the next zombie should be spawned.
   1. Loop 4 \* difficulty times, instantiating new ZombieDog objects with the next spawn location position passed into the constructor.
   2. Add the new objects to the list containing all enemies.
   3. This way the zombies are spread out throughout the map, and when new areas are unlocked, the getNextSpawnLocation() decides which ones should take priority.
3. If an event occurs prompting a boss fight, then spawnBossWave() will be called by the next spawnWave() call.

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

**Unit tests for ensuring map boundaries are inescapable regardless of angle of approach and speed of player:**

1. **Spawn a player object 1 unit in the -x direction from a boundary wall with a large detector object on the other side.**
   1. Input: Vector2 starting from (0.1, 0), increasing each trial by (0.1, 0) to (50, 0)
   2. Input: Vector2 starting from (0.1, 0.1), increasing each trial by (0.1, 0.1) to (50, 50)

The player must not collide with the detector feature, or the trial fails. For each trial, if the detector was not collided with for 0.3 seconds, the trial is considered successful. Trial results will be printed to a file with the format: *success* *forcex, forcey*, where *success* is a 0 if the trial failed, 1 otherwise.

1. **Spawn a player object touching the -x side of a boundary wall with a large detector on the other side.**
   1. Input: Vector2 starting from (0.1, 0), increasing each trial by (0.1, 0) to (50, 0)
   2. Input: Vector2 starting from (0.1, 0.1), increasing each trial by (0, 0.1) to (0.1, 50)
   3. Input: Vector2 starting from (0.1, 0.1), increasing each trial by (0.1, 0.1) to (50, 50)

The player must not collide with the detector feature, or the trial fails. For each trial, if the detector was not collided with for 0.3 seconds, the trial is considered successful. Trial results will be printed to a file with the format: *success* *forcex, forcey*, where *success* is a 0 if the trial failed, 1 otherwise.

1. **Spawn a player object in the center of a boundary wall with a large detector on both sides.**
   1. Input: Vector2 starting from (0.1, 0), increasing each trial by (0.1, 0) to (50, 0)
   2. Input: Vector2 starting from (0.1, 0.1), increasing each trial by (0, 0.1) to (0.1, 50)
   3. Input: Vector2 starting from (0.1, 0.1), increasing each trial by (0.1, 0.1) to (50, 50)

The player must not collide with the detector feature, or the trial fails. For each trial, if the detector was not collided with for 0.3 seconds, the trial is considered successful. Trial results will be printed to a file with the format: *success* *forcex, forcey*, where *success* is a 0 if the trial failed, 1 otherwise.

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

### Work items

|  |  |  |
| --- | --- | --- |
| Task | Duration (Hours) | Predecessor Task(s) |
| 1. GameManager Layout | 4 | - |
| 2. Basic Map Creation | 2 | - |
| 3. Enemy Spawning Algoritm Creation | 6 | 1, 2 |
| 4. Map Aesthetic Refinement | 4 | 2 |
| 5. MapManager Script Creation | 3 | 3 |
| 6. Create Unlockable Areas | 4 | 4, 5 |
| 7. Testing | 4 | 6 |
| 8. Export Minimum Viable Product | 5 | 7 |
| 9. Documentation | 3 | 8 |

### Pert diagram

Diagram

Description automatically generated

### Gantt timeline

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  | 1 | 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  | 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  | 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  |  | 4 | 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 6 |  |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 7 |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 8 |  |  |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |

Blue = Slack, Red = Work Hours