Name\_\_\_\_\_Matt Morrison\_\_\_\_\_\_ Mark \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/50

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

## Brief introduction \_\_/3

My Feature determines enemy movement using either a search pattern for patrolling, or, if it detects the player it can chase until player has reached a minimum safe distance.

## Use case diagram with scenario \_\_14

### Use Case Diagrams

Diagram

Description automatically generated

### Scenarios

**Name:** Enemy Patrol and Chase

**Summary:** The AI will use this function to determine if it will chase the player or continue patrolling

**Actors:** Enemy AI

**Preconditions:** Enemy Unit has been instantiated

**Basic sequence:**

**Step 1:** Check if it can see player

**Step 2:** Move According to exception

**Step 3:** Continuously check if it can see the player

**Step 4:** Calculate and show result.

**Exception-can’t see player:**

**Step 1:** Generate a random search point within a certain distance from the enemy

**Step 2:** Generate path from enemy to search point

**Step 3:** Generate new search point and repeat

**Step 4:** continuously search for player condition ‘can see player’

**Exception-can see player:**

**Step 1:** Lock the ‘raycast’ onto the player

**Step 2:** Move from current position to player position

**Post conditions:** Enemies either continues patrolling or is destroyed

**Priority:** 1

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

### Data Flow Diagrams

**Diagram

Description automatically generated**

### Process Descriptions

Enemy AI:

* Enemy Starts off with creating a path and searching it
* Every update frame check bool function ‘can see player’
* ‘can see player’ returns true or false depending on proximity of player
* If enemy is either following path or chasing player it will move in the vertical/horizontal/diagonal direction

Enemy:

* Enemy when instantiated will have attached sounds, items, and animation movements.
* When the enemy interacts with the player it can take and deal damage until it is dead where it will drop the item.

Diagram 0:

* The user enters the game through the main menu this where the user can also choose to terminate the program
* Upon entering the level the game will determine which level the player is on and environment equal to the difficulty the player is at.
* The player may then interact with the environment and get various types of loot or currency
* The player may also encounter vendors to exchange this currency for other loot items.
* Along with vendors the player may encounter less friendly NPC’s that will attack once in range.
* The NPC’s, Player, and enemies will come from a sprite library as well as an animation collection for the specified sprite
* Lastly once the players animation and sprite have been instantiated a global health/currency monitor will appear on the UI displaying the amount of health and currency remaining

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

[Describe the inputs and outputs of the tests you will run. Ensure you cover all the boundary cases.]

**Example for random number generator feature**

Run feature 1000 times sending output to a file.

The output file will have the following characteristics:

* Max number: 9
* Min number: 0
* Each digit between 0 and 9 appears at least 50 times
* No digit between 0 and 9 appears more than 300 times
* Consider each set of 10 consecutive outputs as a substring of the entire output. No substring may appear more than 3 times.

**Example for divide feature**

|  |  |  |  |
| --- | --- | --- | --- |
| Output | Numerator  (int) | Denominator  (int) | Notes |
| 0.5 | 1 | 2 |  |
| 0.5 | 2 | 3 | We only have 1 bit precision for outputs. Round all values to the nearest .5 |
| 0.0 | 1 | 4 | At the 0.25 mark always round to the nearest whole integer |
| 1.0 | 3 | 4 | At the 0.75 mark always round to the nearest whole integer |
| 255.5 | 5 | 0 | On divide by 0, do not flag an error. Simply return our MAX\_VAL which is 255.5. |

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

[Figure out the tasks required to complete your feature]

Example:

### Work items

|  |  |  |
| --- | --- | --- |
| Task | Duration (PWks) | Predecessor Task(s) |
| 1. Requirements Collection | 5 | - |
| 2. Screen Design | 6 | 1 |
| 3. Report Design | 6 | 1 |
| 4. Database Construction | 2 | 2, 3 |
| 5. User Documentation | 6 | 4 |
| 6. Programming | 5 | 4 |
| 7. Testing | 3 | 6 |
| 8. Installation | 1 | 5, 7 |

### Pert diagram



### Gantt timeline

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