GADE7311 POE Concept Document

Table of Contents

[**High Concept Statement**](#_6nscwv949odh) **3**

[Premise](#_xqlrkspv07wy) 3

[Intended Audience](#_im4lx9jdsxt) 3

[Genre](#_k0fz3mx2452t) 3

[**Game Rules**](#_h9f9hrd39dic) **4**

[**Game State Representation**](#_ed8vk1r3rm3x) **5**

[**Game State Utility Function**](#_okqqednbpzc4) **6**

# High Concept Statement

## Premise

The premise of this game follows a chess-like formula but the pieces don’t have the same form of movement and it takes place over a grid rather than a black and white board. Each side has a \*tbh colour\* and their goals are to annihilate each other’s pieces. Each piece has different roles and different varying HP’s as well as damage, each piece moves differently on the grid.

## Intended Audience

The intended audience will be people who enjoy a mixture of turn-based strategy and action combat in a game. The ideal age for the audience would be between 14 and 20, but the game would be able to cater to anyone that has an interest in the genre that the game would be based in. Based on Bartle’s player types the target audience would most likely fall into the Killer and Achiever categories.

## Genre

The genre of the game would be:

* Turn-based
* Action
* Strategy
* Sci-fi
* Post-apocalyptic

# Game Rules

1. Each player can make 1 move a turn
2. Each player has 1 robot in their possession as well as other robots to act as support bots. Players can have up to 3 support bots.
   1. These support bots can have a variety of classes:
      1. Guard - Nullifies damage
      2. Attack - damages HP of opponent
      3. Techy - Repairs damage to the bot
3. Each robot has an HP that can be reduced by the opposing player through attacks
4. Once a robot’s HP reaches zero the battle is over.

# Game State Representation

For the main menu I decided to use the scene manager to swap from the main menu into the main game so that it will allow for easier managing of the scenes and as well as a simpler way to manage the scenes so that they will go in the selected order that I choose them to go in. What I did in the code was implement the branch of UnityEngine.SceneManagement and I made a public void which will allow me to load in 1 different scene at a time.

We decided to go with the idea of a 1v1 situation in the game because it will allow for an easy experience for us as inexperienced game developers. The main board of the game will have a grid like design with blocks that are going to be randomly generated maps with some hazardous obstacles that if a player steps on it they will take some damage. There will also be some covers that we will develop later into the game’s progress so that it will add a bit more variety to the game play where the players will be able to take cover behind and get nulled from all damage they will receive for 1 or 2 rounds before the cover will be destroyed.

Turn-based system was implemented through the use of counter variables.

In order to ensure that both players had their chance before the turn ends a 2nd variable was established. This 2nd variable names playerCounter would vary between 1 and 2. Once it reaches 2 it would reset back to 1 and increment the turn counter, thus moving to the next turn.

If it is not the players turn the program would look to the playerCounter variable in order to establish who’s buttons must be non-interactable.

We used the health bars to be sliders because it would be better off than just using an amount of lives left because it would’ve been harder to input that way. We made the HP into a variable in the main controller for the character. We put it into a void called DamageOpponent() and we linked the enemy robot by the amount of damage that they will be taking called “enemyRobot.RoboCurrentHealth -= RoboAttackStrength;”. Then we linked it with a script called HealthBarController where we wrote it to find the variables from the player script and to how much health the player has left out of 100 it will determine how full the slider’s bar will remain.

The buttons’s script conjoins with the methods of the turn controller script in order for the players to not just spam the attack button until either one of them dies first so we added the IncrementTurn() method and mapped it to each of the buttons so that once the button is clicked the players are going to have to wait for the other player to finish their turns before they are able to make their next move. The way the code is implemented is through the line turnController.IncrementTurn(); under each of the methods.

# Game State Utility Function

The game will start out with the map randomly generated to incorporate hazardous or beneficial tiles upon which the player can stand. The AI will try and avoid these hazardous tiles if possible in order to mitigate any damage done upon it. (This will be implemented later on in the program.)

Next the AI will try and find the safest route, buff tiles prioritised, towards the player in order to get within range and deal damage.

Should the AI be out of cover it will shield itself from an attack in order to prevent a first strike from a player if it was not within attacking range.

If the AI was indeed within attacking range then it will use its attack to deal damage.

Should the AI take sufficient damage then it will use its heal action in order to place it on cooldown so it can be used again later.

If one of the players win then the victor’s name will be displayed where the turn counter was.