Gesture Based UI Development Main Project

Aaron Moran – (G00356519)

GitHub: https://github.com/Moran98/pokemon-battle-sequence

Table of Contents

Introduction

We have been tasked with creating a Gesture Based game using either Hardware or Voice Controls to perform the functionality of the game. I have decided to create a clone and tweak of the battle sequence in the Pokémon game series. I have used what I have learned over the past year to incorporate the Grammar Recognizer functionality and applied a State Machine so that it determines the sequences throughout the battle.

The game has a character battle selection that is completely controlled by voice commands which then will load the enemy's Pokémon and you have the option to either Attack / Heal / Flee. Each Player/Opponent has a defined Damage score and Health score. Once you have performed an action the state will change to 'ENEMYSTURN'. Depending on the battles outcome the state will either change to 'WIN' or 'LOSS'.

Purpose of the application

The main purpose of this application is to demonstrate the use of voice commands in a state defined battle sequence. The goal of the game is to defeat your opponent in battle using attacks. The game has been built through the Unity engine which makes developing a game in 2D simple and effective.

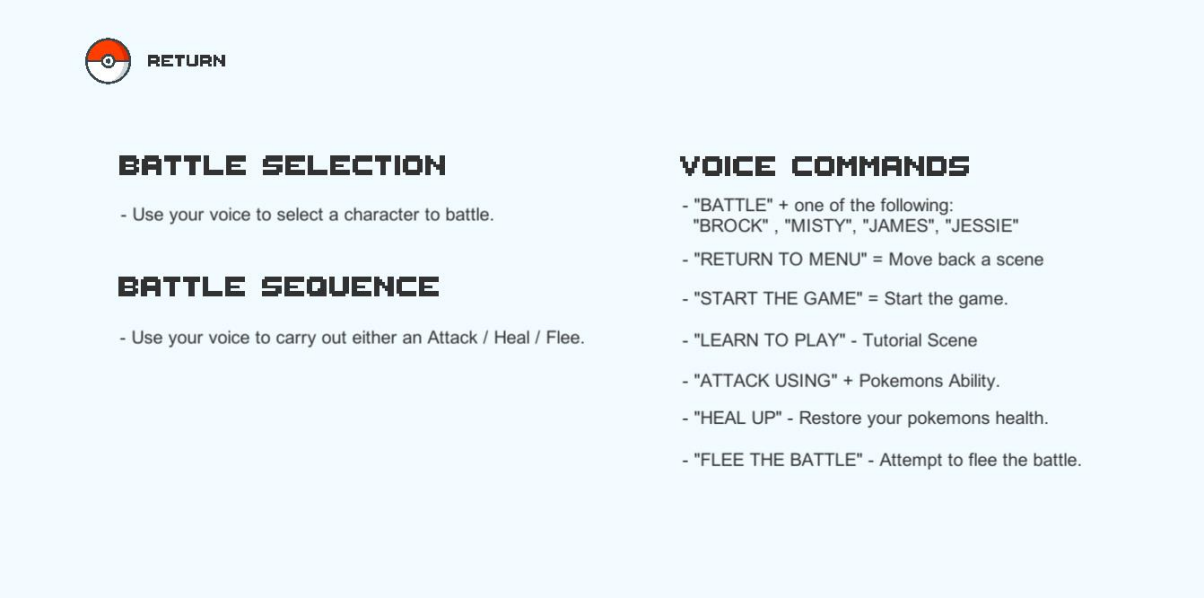
Main Menu

In the image below this is the screen that the user will be greeted with. The player can choose either to start the game or learn the games mechanics and commands. Either option is controlled by voice commands.

(Fig 1.1. Main Menu)

Instructions

The Instructions scene consists of the voice commands and instructions for the Battle Selection scene and the Battle Sequence scene. The commands are basic as it means the game is easy to play which makes the user experience smooth and easy to play.



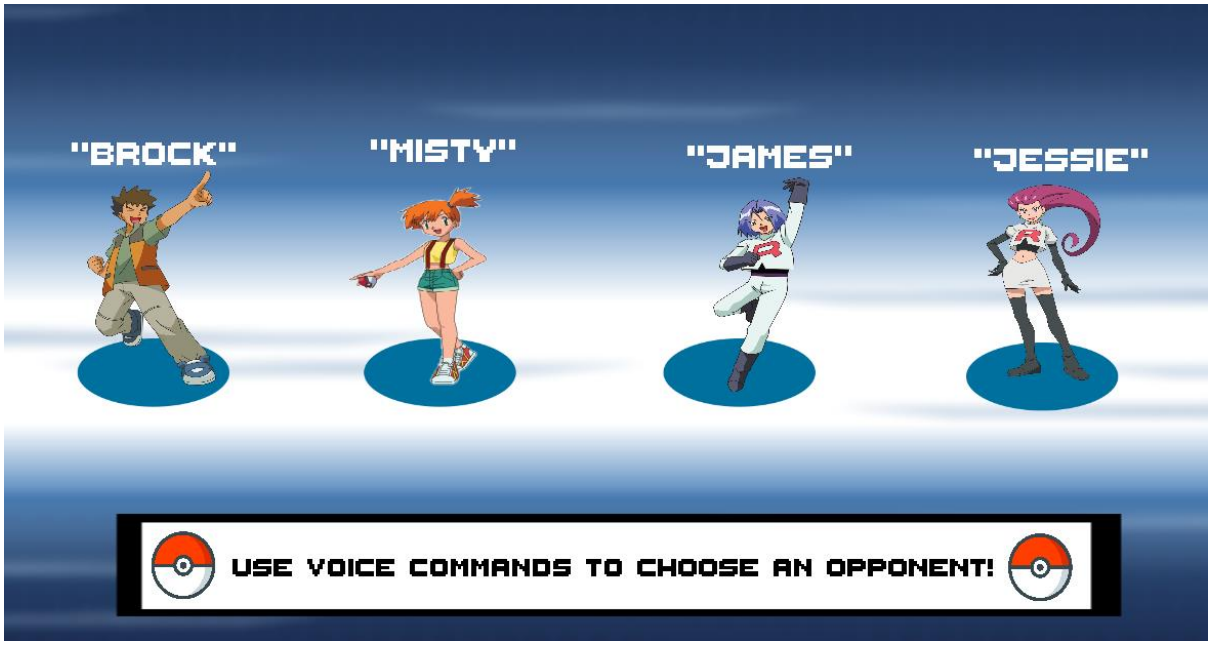
(Fig 1.2. Instructions Scene)

Battle Selection

In the image below you can see that there are four opponents to choose from to battle. Each opponent has a different Pokémon who has different health and damage attributes. You must use voice commands to choose who to battle. Once an opponent has been chosen the battle will commence.

Voice Commands:

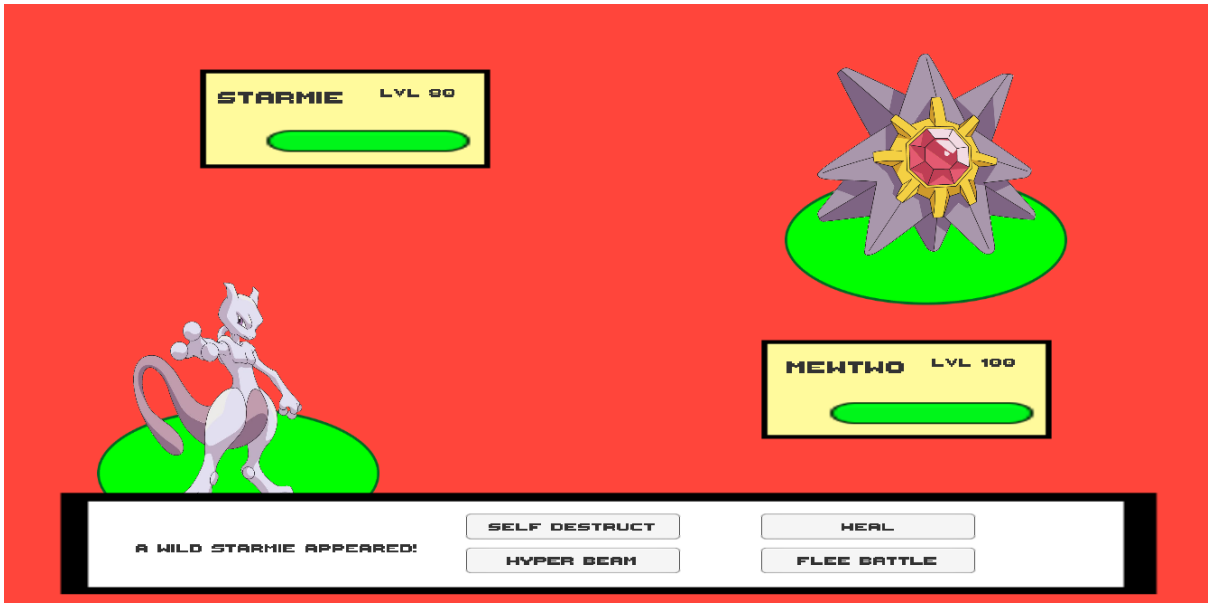
- “Battle” + opponent’s name (e.g. “Battle Brock”).



(Fig 1.3. Battle Selection Scene)

Battle Sequence

- Once the battle begins you will be prompted to choose an option between – ‘Attack using’ + ‘HYPER BEAM’ or ‘SELF DESTRUCT, ‘HEAL UP’ or ‘FLEE THE BATTLE’. Once you have chosen an action whether it be attacking or healing the action will be performed and the state will change to the opponents turn.
- The attack ‘Hyper Beam’ is controlled with a random number generator, if the number lands between 7-10 then the attack will fail. This is to implement some randomness into the game, so it is not just attacking back and forth.
- If you choose to ‘FLEE THE BATTLE’ this will select a random number between 1-10 and if the number is greater than five and less than or equals to ten you will be unable to flee. Once your opponents Pokémon reaches zero health the state will change to ‘WIN’. If your Pokémon’s health reaches zero first the state will change to ‘LOSS’.



(Fig 2.1. Battle Sequence)

Gestures identified as appropriate for this application

I researched the use of VR for the implementation of this game style and the possibilities are quite impressive. The ability to use the VR headset to interact with the battle and choosing attacks with hand gestures would make the experience a lot more intuitive. Unfortunately, I could not test this feature out as VR headsets are expensive and could not utilise the opportunity to test out this design.

Also, using the Kinect would open possibilities to recognize hand movements to navigate around the UI of the game but faces difficulties when prompting the character to perform an action. By implementing Voice Controls, this was the easiest and most user-friendly way for the player to tell their character what action to perform, this could not be achieved through hand gestures.

Hardware used in creating the application

For this application the only feature that I have implemented and utilized are the Voice Recognition controls, so the only hardware that is being used is the Microphone. I have a set file of grammar which is used throughout the game and it is loaded once the game has started.

Voice Commands

- By using voice commands, I have a switch statement setup to perform the actions depending on the recognized phrase. Each of these methods will carry out a function and then change the games state. Below is an example of the battle sequence phrases.

```
private void Commands(){
    switch (valueString)
    {
        case "ATTACK USING SELF DESTRUCT":
            onAttackButton();
            valueString = "";
            break;
        case "ATTACK USING HYPER BEAM":
            onHyperBeam();
            valueString = "";
            break;
        case "HEAL UP":
            onHealingButton();
            valueString = "";
            break;
        case "FLEE THE BATTLE":
            onFleeingButton();
            valueString = "";
            break;
        default:
            break;
    }
}
```

(Fig 3.1. Switch statement using the Grammar Recognizer to perform actions.)

- Once the game has started the Grammar is loaded into the application. Along with the Grammar loading in, the state of the game is set, and the battle gets configured.

```
private void Start()
{
    gr = new GrammarRecognizer(Path.Combine(Application.streamingAssetsPath,
                                            "SimpleGrammar.xml"),
                               ConfidenceLevel.Low);
    Debug.Log("Grammar loaded!");
    gr.OnPhraseRecognized += GR_OnPhraseRecognized;
    gr.Start();
    if (gr.IsRunning) Debug.Log("Recogniser running");

    state = BattleState.START;
    StartCoroutine(SetupBattle());
}
```

(Fig 3.2. Loading in the Grammar file and setting up the battle.)

Architecture for the solution

- For the design of my project I have placed all the battle actions into a file named ‘BattleSystem’. Inside this file I have defined the states which will happen throughout the game depending on the commands that are spoken. The states control the flow of the game and determine a winner.

```
public enum BattleState { START, PLAYERTURN, ENEMYTURN, WON, LOST}
```

(Fig 4.1. Setting the states.)

- When the battle sequence begins, I am instantiating the player and enemy prefabs depending on which opponent is chosen to battle. The correct enemy HUD is displayed followed by the state changing to allow the battle to begin.

```
IEnumerator SetupBattle()
{
    playerUnit = Instantiate(playerPrefab, playerBattleStation).GetComponent<Unit>();
    enemyUnit = Instantiate(enemyPrefab, enemyBattleStation).GetComponent<Unit>();

    dialogText.text = "A wild " + enemyUnit.unitName + " appeared!";

    playerHUD.SetHUD(playerUnit);
    enemyHUD.SetHUD(enemyUnit);

    yield return new WaitForSeconds(2f);

    state =  BattleState.PLAYERTURN;
    PlayerTurn();

}
```

(Fig 4.1. Setting up the battle.)

- By implementing a state machine, this allows for the performance and execution between actions to transition smoothly and without error, once the state has been changed you cannot perform any actions during the Enemy's turn. Below is an example of the attacking sequence and where the state will change given the enemy's health has not reached zero.

```
IEnumerator PlayerAttack()
{
    // Damage the enemy
    bool isDead = enemyUnit.TakeDamage(playerUnit.damage);

    enemyHUD.SetHP(enemyUnit.currentHP);
    dialogText.text = "The attack is successful!";
    Debug.Log("Successful attack");

    yield return new WaitForSeconds(2f);

    if(isDead)
    {
        // End
        state = BattleState.WON;
        EndBattle();
    } else{
        // Enemy turn
        state = BattleState.ENEMYTURN;
        StartCoroutine(EnemyTurn());
    }
}
```

(Fig 4.1. Changing the states when an action is called.)

The only libraries which I have utilised in this project are the Speech Recognition and UI as the main features which are used are voice commands and text to display the changing of states.

Conclusions & Recommendations

Overall, I am happy with the outcome of this project as I utilised material that I was taught over the past year between Voice Recognition and State Machines. From this project I have learned that making things complicated will only slow down the development process, so flattening my idea out to its most basic components with the implementation of Voice Controls to navigate and perform actions in the least amount of commands was the best way to achieve a smooth and enjoyable user experience.

Our previous assignment on a Finite State Machine helped me to complete this game as it gave me an insight to its uses and the game functionality that can be achieved with it. By implementing the state machine and aligning them with voice commands the game was enjoyable to develop and to play.