Project Title: Pokemon Battle Platform

Team Members: Zack McClary, Haihan Yu, Yuwei Cai, Dale Song

UT EIDs: ZRM276, YH5875, YC7948, DS38368

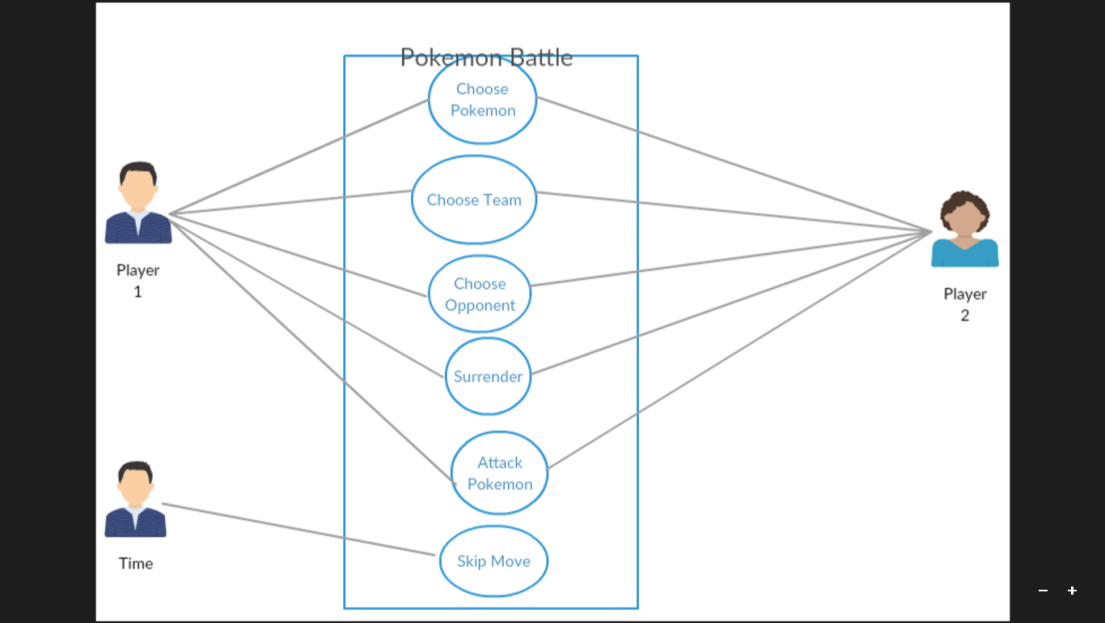
Repository URL: <https://github.com/yc1000/461GroupProject>

**Motivation**

Pokemon is a popular game among people of all ages. The battling mechanics involve luck, strategy, and fun. However, a user may just want to battle and skip the arduous training process. We aim to eliminate all these deficiencies and give an opportunity for pokemon fans to enjoy the fun of a pokemon fight.

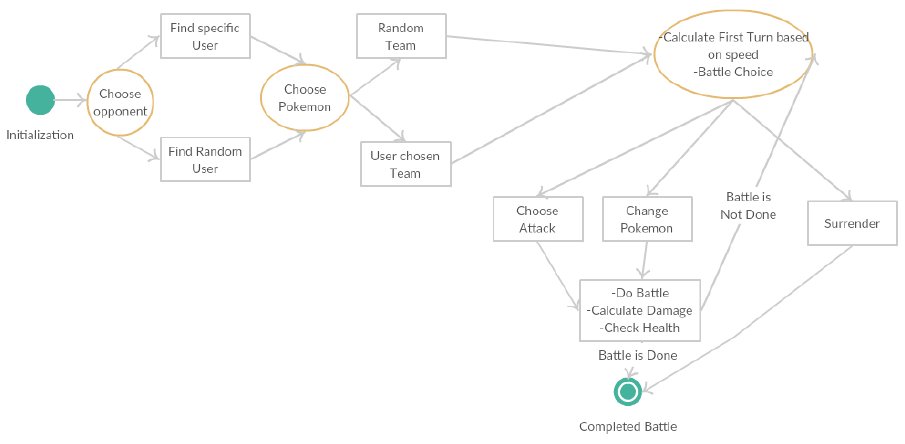
We will set up an online pokemon fighting platform that enables users to create a team and fight other players. In the platform, the users will have access to a wide selection of pokemon that they can select to battle with. Users can also choose to battle with a randomly generated team. Our application will connect users to their opponents. These opponents can be either online friends or random challengers. This way, users can avoid the long process of catching and training pokemon and skip straight to the fun.

**Feature Description and Requirements**

PokePVP will be a turned based fighting game. The main objective of the application is to make all of the pokemon on the opponent’s team ‘faint’. 

A pokemon will faint once it’s Health Points (HP) reach zero. For clarification, we have included user stories: As a player I want to choose the best pokemon team that will help me defeat my opponent; as a player, I want to select the best attack move so that my opponent’s pokemon’s Health Points lowers as much as possible; As a player, I want to choose my opponent so that I can play with my friends or random people without having to go through the process of catching and training the pokemon I want.

The state chart diagram above describes the general process of our proposed app. The player chooses an opponent. Then both players assemble their respective pokemon teams from our database of pokemon. Next, it’s time to fight. The player has three choices: choose an attack, switch your pokemon, and surrender. If the player selects surrender, he/she will automatically lose the game. The battle is finished when either player’s team of pokemon has ‘fainted’.

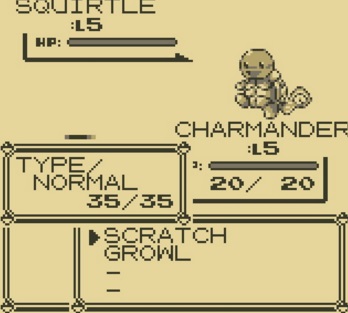


The image below on the left shows the state when an opponent has been found. Each player will have a unique ID (in this case, our opponent’s ID is BLUE).



The image on the right shows the player’s battle options of fighting, switching, and surrendering.

If we choose to fight, we have the option to select the attack we want to use. Each attack has damage and/or a special effect. Our goal is to make the enemy Squirtle faint before our beloved Charmander. In this scenario, we only have one pokemon on our team. This means that if our Charmander faints first, we lose the battle. However, if we had multiple pokemon on our team, we could choose which pokemon battles our opponent next. Since we are the first to choose a battle option, we must wait for our opponent’s attack choice after we have selected ours.

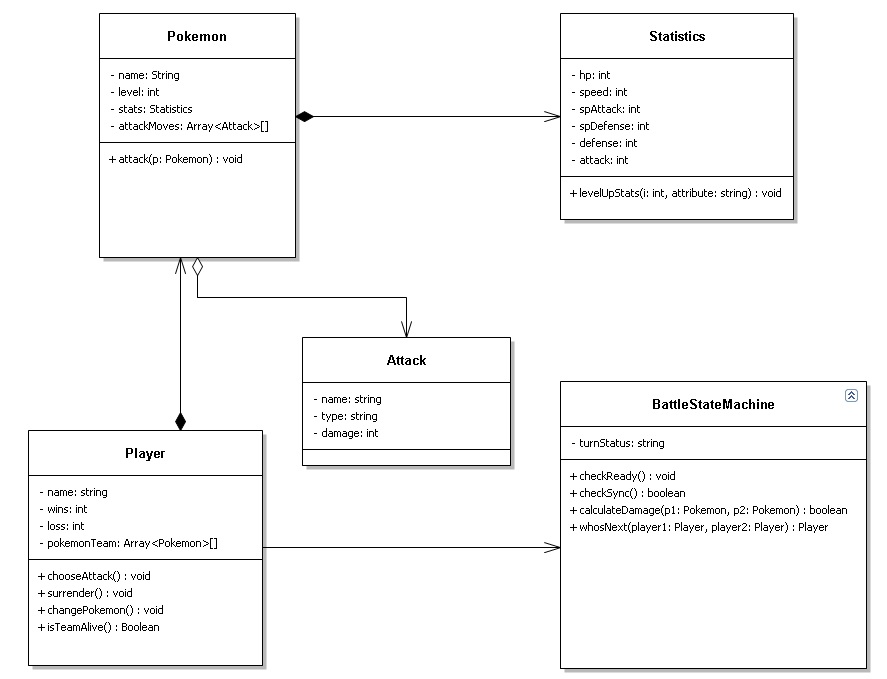


The game continues in this format. The first player selects an option then waits for the opponent to choose. After both players have selected an option, damage and health is calculated (if Fight was chosen) or pokemon are switched (if Change Pokemon was chosen). The game ends when either player’s pokemon team has fainted.

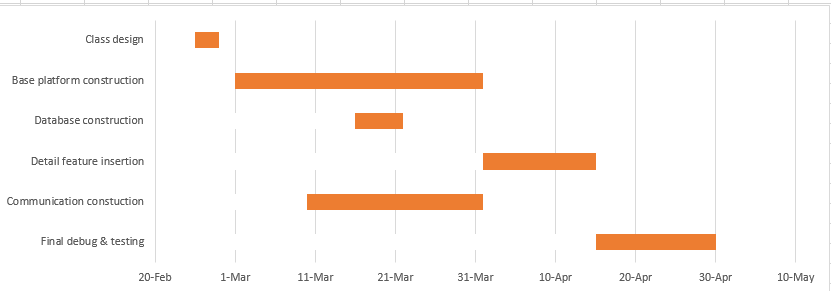
In order for PokePVP to be enjoyable, our app will require low latency between players’ responses. A player must see his opponent’s response within three seconds. Our app must also require an intuitive user interface to promote ease of use and smooth functionality in all aspects of the game. PokePVP will require no more than one gigabyte of memory.

**Design Document**

The five classes we will be using are our Pokémon, Player, Attack, Statistics, and Battle State Machine classes. The Player class will store 3-6 Pokémon and information needed to keep track of the battle progress. The Pokémon class will contain the conditions of the pokemon in battle. The attack class will hold the battle choice information, which will be passed between the devices. The statistics class will hold attributes for the pokemon that can be accessed when calculating damage for attacks. The Battle State Machine class will be a state machine that handles each battle sequence.



The technologies we will be implementing include but are not limited to – displaying sprites of Pokémon, animations for attacks, and sending and receiving information between phones. The complexity of displaying images will be dependent on how we plan to store images. One possibility is choosing to download images from online each time we add a sprite to the UI. This would save us a lot of memory, however it will use up more data for downloads and can lead to more issues if there is a bad connection. The other way to display images is as a bitmap that is stored locally. This would take up a lot of storage and would require more research on how to best store each bitmap efficiently. Animations can be as complex or complicated as we choose to make them. There are endless amounts of public libraries for downloading animations on the internet. Our most complicated process will involve communication between android devices. The two simplest ways of doing this would be using either Bluetooth or TCP/UDP IP connections. Bluetooth would work well because there is no need for a central server. It is limited however, in that the two devices will have to be within 20 feet of each other. The other option is to use TCP/IP connections just like a computer. One of the phones will act as a server. However, issues could arise if the IP address of a phone is not globally reachable. Because of the amount of complications of the connection process, this is where most of time for testing will be spent. We will have to start with simple things like a string or integer and work our way to sending battle choices. We must also test with different models of phones along with different internet connections (Wi-Fi, 3G, 4G, etc.). The other technologies, animations and sprites, can be tested locally. We have included a Gantt chart outlining a preliminary schedule for our project.



**Feasibility**

The main factors that may cause our project to fail are the lack of familiarity with Android Studio and the fact we are counting on using some pre-existing code that may not be reliable. Out of us four, only Zack has had experience developing an Android app. That means we will need to set aside time to familiarize ourselves with Android Studio. Our other obstacle is that our design includes communication modules, and our team is not familiar with phone to phone communication. We will look into borrowing code for the connection and that code must be tested. We could end up having to restart the project if the code is not up to par. To help mitigate that risk we will seek out multiple sources of code and experiment with each one to find the best source for our needs. We will also try to utilize modular programming methods so that we can switch out code if needed. Other minor concerns include that we may have too many features. However, we can choose to leave out minor details that Pokémon includes. Our core set of features are the inter-device communication and basic battling. Overall, if we take proper risk reduction procedures, we believe we will finish the project in time.