# ICS Summative: Pokemon Emerald Elite Four Battle

David Chen-Li
Bing Li
Nicholas Vadivelu

Our ICS summative replicates one of the final battles in the game Pokemon: Emerald Version. Taking into consideration all aspects of the original game, we attempted to accurately recreate a similar version with our knowledge of Java and its resources.

Pokemon is centered around capturable creatures known as “Pokemon.” These creatures are captured by humans and trained to fight each other in non-lethal battles for sport. How effectively Pokemon can fight is up to the trainer who owns them.

The game Pokemon has many different components to it that make it such a unique game. These components include:

- Pokemon Types
- Move Types
- Effectiveness/weaknesses
- Status afflictions
- Levels
- Statistics

Each of these components plays a specific role in determining the most efficient way of playing Pokemon. The player must take into consideration each one before selecting which Pokemon to use, which moves to use, and in what order does he/she use them. The game uses these components for calculations in battle mechanics, such as damage, health, and status.

## Pokemon Types:


This is what determines the Pokemon’s abilities, weaknesses, and effectivenesses. Pokemon can have at most a combination of two types. Using these types, Pokemon have a resistance and a weakness to other types that affect their type (E.g Fire type is weak to Water type). In our program, we assigned each type to an integer variable in our Pokemon class.

```
// Constructor - Creating a new pokemon, if creating single type, always input 0 as t2
public Pokemon (String n, int l, int h, int a, int d, int sa, int sd, int s, int t1, int t2)
```

```
/*
 Types are represented by integers, the legend is as shown:
 0 - None            9 - Ground
 1 - Normal          10 - Psychic
 2 - Fire            11 - Rock
 3 - Fighting        12 - Ice
 4 - Water           13 - Bug
 5 - Flying          14 - Dragon
 6 - Grass           15 - Ghost
 7 - Poison          16 - Dark
 8 - Electric        17 - Steel
```

Using the constructor pictured on the last page , we assigned type values to the variables **int t1** and **int t2.** If the Pokemon only had one type, then the **int t2** variable would be set to 0. We created a legend to differentiate which integers we assigned to the different types, and we followed the chart above when coding the information about Pokemon types.

**Move Types:**

Depending on the Pokemon's type, the moves that the Pokemon will be able to learn will primarily compose of moves from the same type as the Pokemon. There are also limitations; the number of moves and the kind of moves that a Pokemon can learn is unique to each individual Pokemon. Unlike Pokemon types, Move types are limited to only one type. Moves can also inflict status conditions on the target Pokemon, and affect stats of either Pokemon on the field.

**Statistics:**

Each Pokemon has its own unique set of statistics that determine the Pokemon's level, HP (Hit points), defense, attack, special attack, special defense, and speed. These statistics are used in Damage Calculation, and affect how much damage the Pokemon can deal and take. As Pokemon is a turn based game, the Speed of the Pokemon affects which Pokemon goes first in a battle.

There are also statistics for Moves that Pokemon use. These include the category of the move, (Physical, Special, Status), the Type of the move, the Power of the move, or how much damage it does, Accuracy, and the number of Power Points it has. These statistics specifically affect how much damage the move does to a defending Pokemon.

| Swampert L60 | | 0 EV | |
|---|---|---|---|
| | | Min | Max |
| HP | | 190 | 208 |
| Atk | = | 137 | 155 |
| Def | = | 113 | 131 |
| SpAtk | = | 107 | 125 |
| SpDef | = | 113 | 131 |
| Spd | = | 77 | 95 |

This is a chart showing an estimate of the stats that a level 60 Swampert would have. We took similar information for all the other Pokemon used in the game, and constructed arrays of Pokemon for each opponent and for the player. Within the array of 5-6 Pokemon, we created

another array for each Pokemon in order to allocate memory for the moves that the Pokemon has.

```
Player [0] = new Pokemon ("Swampert", 60, 190, 137, 113, 107, 113, 77, 4, 9, "Frames\\Player\\swampert.gif");
```

This is an example of the code we used for Swampert, which is controlled by the player. This allowed us to declare the name of the Pokemon, in this case Swampert, and the stats that a level 60 Swampert would usually have. We used a Pokemon class which we created earlier in order to allocate memory for the statistics. The stats were inputted in this order: Level, HP, Attack, Defense, Special Attack, Special Defense, Speed, Type 1, Type 2.

```
Player [0].move [0] = new Move ("Earthquake", 9, 1, 100, 100, 10); //No effect
Player [0].move [1] = new Move ("Hyper Beam", 1, 2, 120, 90, 5); //No effect
Player [0].move [2] = new Move ("Hammer Arm", 3, 1, 90, 90, 10); //No effect
Player [0].move [3] = new Move ("Surf", 4, 2, 90, 100, 15); //No effect
```

This array is associated with the Swampert code, and uses methods in the Move class in our program to allocate memory for the statistics of the move. The array is used to identify the four moves that Swampert can use, and is used by our damage calculation method to calculate how much damage Swampert can do.

**Levels:**

Levels in Pokemon determine the strength of the Pokemon. The level of the Pokemon is based on how much experience the Pokemon has gained. Leveling up a Pokemon increases their stats by a small amount, and may allow the Pokemon to evolve or learn new moves depending on the level.

**Status Afflictions:**

These affect how well the Pokemon can battle. In our game, we incorporated five non-volatile statuses that affect the player's Pokemon. These status conditions cannot be removed until the player has beaten the opponent. The effects of these statuses range from making the afflicted Pokemon take damage over time to completely incapacitating the Pokemon until it is fully restored at the end of the battle. The statuses and effects are:

- Burn
  - Physical damage dealt by afflicted Pokemon is reduced by ½
  - Afflicted Pokemon loses ⅛ of its maximum HP every turn
- Freeze
  - Causes afflicted Pokemon to become unable to make a move
  - Has a 20% chance to be cured on its own
- Paralyze
  - Afflicted Pokemon is unable to attack for 25% of the time
  - Pokemon speed is reduced to 25%
- Poison
  - Causes afflicted Pokemon to lose ⅛ of its maximum health every turn
- Sleep
  - Completely incapacitates the afflicted Pokemon

## Damage Calculation:

In the game, damage calculations are made using the damage formula. This formula takes into account multiple factors and statistics about both the attacking Pokemon and the defending Pokemon in order to calculate the amount of damage the affected Pokemon takes.

$$Damage = \left(\frac{2 \times Level + 10}{250} \times \frac{Attack}{Defense} \times Base + 2\right) \times Modifier$$

The amount of damage your Pokemon deals is based on:

- The level of your Pokemon
- The attack statistics of your Pokemon
- The defence statistics of your **opponent's** Pokemon
- The base attack statistics of your Pokemon's moves
- Modifier

Level, attack, defence, and base numbers are already there for the calculation; they're numbers that do not change throughout the course of the battle, unless a move is used to specifically change those stats for the duration of the battle. The attack and defense values change to Special Attack and Special Defense if the move used is not a physical move.

The value for Modifier is calculated using this formula:

$$Modifier = STAB \times Type \times Critical \times other \times (random \in [0.85, 1])$$

| Attacking type \ Defending type | NORMAL | FIGHT | FLYING | POISON | GROUND | ROCK | BUG | GHOST | STEEL | FIRE | WATER | GRASS | ELECTR | PSYCHC | ICE | DRAGON | DARK |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NORMAL | 1× | 1× | 1× | 1× | 1× | ½× | 1× | 0× | ½× | 1× | 1× | 1× | 1× | 1× | 1× | 1× | 1× |
| FIGHT | 2× | 1× | ½× | ½× | 1× | 2× | ½× | 0× | 2× | 1× | 1× | 1× | 1× | ½× | 2× | 1× | 2× |
| FLYING | 1× | 2× | 1× | 1× | 1× | ½× | 2× | 1× | ½× | 1× | 1× | 2× | ½× | 1× | 1× | 1× | 1× |
| POISON | 1× | 1× | 1× | ½× | ½× | ½× | 1× | ½× | 0× | 1× | 1× | 2× | 1× | 1× | 1× | 1× | 1× |
| GROUND | 1× | 1× | 0× | 2× | 1× | 2× | ½× | 1× | 2× | 2× | 1× | ½× | 2× | 1× | 1× | 1× | 1× |
| ROCK | 1× | ½× | 2× | 1× | ½× | 1× | 2× | 1× | ½× | 2× | 1× | 1× | 1× | 1× | 2× | 1× | 1× |
| BUG | 1× | ½× | ½× | ½× | 1× | 1× | 1× | ½× | ½× | ½× | 1× | 2× | 1× | 2× | 1× | 1× | 2× |
| GHOST | 0× | 1× | 1× | 1× | 1× | 1× | 1× | 2× | ½× | 1× | 1× | 1× | 1× | 2× | 1× | 1× | ½× |
| STEEL | 1× | 1× | 1× | 1× | 1× | 2× | 1× | 1× | ½× | ½× | ½× | 1× | ½× | 1× | 2× | 1× | 1× |
| FIRE | 1× | 1× | 1× | 1× | 1× | ½× | 2× | 1× | 2× | ½× | ½× | 2× | 1× | 1× | 2× | ½× | 1× |
| WATER | 1× | 1× | 1× | 1× | 2× | 2× | 1× | 1× | 1× | 2× | ½× | ½× | 1× | 1× | 1× | ½× | 1× |
| GRASS | 1× | 1× | ½× | ½× | 2× | 2× | ½× | 1× | ½× | ½× | 2× | ½× | 1× | 1× | 1× | ½× | 1× |
| ELECTR | 1× | 1× | 2× | 1× | 0× | 1× | 1× | 1× | 1× | 1× | 2× | ½× | ½× | 1× | 1× | ½× | 1× |
| PSYCHC | 1× | 2× | 1× | 2× | 1× | 1× | 1× | 1× | ½× | 1× | 1× | 1× | 1× | ½× | 1× | 1× | 0× |
| ICE | 1× | 1× | 2× | 1× | 2× | 1× | 1× | 1× | ½× | ½× | ½× | 2× | 1× | 1× | ½× | 2× | 1× |
| DRAGON | 1× | 1× | 1× | 1× | 1× | 1× | 1× | 1× | ½× | 1× | 1× | 1× | 1× | 1× | 1× | 2× | 1× |
| DARK | 1× | ½× | 1× | 1× | 1× | 1× | 1× | 2× | ½× | 1× | 1× | 1× | 1× | 2× | 1× | 1× | ½× |

These matchups are suitable for Generation II to Generation V.

STAB stands for Same Type Attack Bonus, which provides a damage boost if the user of the move is of the same type as the move. This boost is a 50% increase in damage from the attacker, which results in a multiplier of 1.5.

Type is used to calculated the bonus damage or damage reduction depending on the types of the Pokemon. For example, as Fire type Pokemon are weak towards Water type Pokemon, the Type parameter in the equation would be 2. In the case of Pokemon with two types, if there is a double

weakness towards a move then the Type parameter would be 4 (Eg. a Bug/Steel type Pokemon that is damaged by a Fire type move).

Critical is used to calculate the damage multiplier from a critical hit during the game. A critical hit is a random event in the game, and results in a multiplier of 2.

The Other category is used to determine the effects that items, inherent abilities, and terrain advantages have on the damage of the attacking Pokemon.

All of these factors are then multiplied by a random number from 0.85 to 1 in order to calculate the Modifier.

In our game, we decided to remove the Critical Strike mechanism from the Modifier calculation, but retained the rest of the formula. We used loops and our classes for Pokemon and Moves. Using these we created a method that would automatically run through the calculations, and change the output display accordingly.

This is the loop we used to calculate the Type Effectiveness:

```
// Type effectiveness multiplier
        double multiplier = 1;
        for (int i = 0 ; i < defender.type.length ; i++)
        {
           multiplier *= weakness [move.type] [defender.type [i]];
        }
```

This the loop for STAB.:

```
// STAB (Same type attack bonus)
        for (int i = 0 ; i < attacker.type.length ; i++)
           if (attacker.type [i] == move.type)
              multiplier *= 1.5;
```

This is the code for the calculation of the random number between 0.85 and 1:

```
// random variable between .85 and 1
        multiplier = multiplier * ((int) (Math.random () * 15) + 85) / 100;
```

This is the code for damage calculation based on whether or not the move is a physical or special attack. :

```
int damage;
           if (move.category == 1)
              damage = (int) ((((2 * (double) attacker.level + 10) / 250) * (((double) attacker.atk * mod [attacker.stage [1]] /
100) / ((double) defender.def * mod [defender.stage [2]] / 100)) * move.power + 2) * multiplier);
           else
/ 100) / ((double) defender.spdef * mod [defender.stage [4]] / 100)) * move.power + 2) * multiplier);
```

There are multiple steps involved in calculating the damage done in our program. See the attached flowcharts.

## Variable Declarations:

```
static Console c;          // The output console
   static BufferedImage image, tex;
   static Graphics2D g, t;
   static Image background, upback;
```

```
    static Poke first;
    // Global variables so methods have easy access
    public static AudioPlayer Music[] = {new AudioPlayer ("Resources\\Audio Resources\\Elite Four.mp3"), new AudioPlayer
("Resources\\Audio Resources\\Champion.mp3") };
    public static Pokemon Opponent[];
    public static String oppName[] = {"Sidney", "Phoebe", "Glacia", "Drake", "Wallace"};
    public static Pokemon Player[] = new Pokemon [6];
    public static int p; // Player's current pokemon
    //public static Image Sid,Pho,Gla,Dra,Wal,Pla; // Gifs for images of the league and player
    public static JFrame f;
    private static JButton[] movess = new JButton [4]; //holds moves
    private static JButton[] pokemonn = new JButton [6]; //holds pokemon
    private static JButton attack, swich; //options//quit, cancel
    static ImageIcon userPokes[] = new ImageIcon [6];
    static ImageIcon userPokes2[] = new ImageIcon [6];
    private static int textdelay = 30, usehp, opphp, defendhp;
    static Color opp, use, bx = new Color (231, 231, 231);
    static AudioPlayer attacksounds[] = {new AudioPlayer ("Resources\\Audio Resources\\nuller.mp3"), new AudioPlayer
("Resources\\Audio Resources\\not_effective.mp3"), new AudioPlayer ("Resources\\Audio Resources\\normaldamage.mp3"),
null, new AudioPlayer ("Resources\\Audio Resources\\super_effective.mp3")};  //, new AudioPlayer ("Resources\\Audio
Resources\\nuller.mp3"), new AudioPlayer ("Resources\\Audio Resources\\nuller.mp3"), new AudioPlayer ("Resources\\Audio
Resources\\normaldamage.mp3"), new AudioPlayer ("Resources\\Audio Resources\\normaldamage.mp3")};
      static AudioPlayer statuser = new AudioPlayer ("Resources\\Audio Resources\\effect_recover2.mp3");
      static AudioPlayer turnsound = attacksounds [2];
```

## Gameplay:

At the beginning of our game, the player has reached a point where he is able to challenge the best Pokemon trainers in the region, also known as the Pokemon League, which is comprised of the Elite Four and the Champion. The player comes with a team of six Pokemon, and must use these six Pokemon to defeat the entire Pokemon League.

As each of the Elite Four have a Type that their Pokemon team generally conforms to, the Player must take this into account as he/she plays the game. After successfully defeating a League member, the player's Pokemon are automatically restored to their full health and PP, and all Status Conditions are removed. This is normally not the case in the original Pokemon game, but as we removed the use of Items in our program, we decided this would be a fair replacement.

## Issues Encountered:

- Error in Weakness/Resistance Array
    - We discovered an unusual damage output on Elite Four Phoebe's Pokemon from Pokemon that use the move Surf on the player's team. This was caused by accidentally typing a comma instead of a period in our Weakness/Resistance array, and raised the multiplier from 0.5 to 5.
- HP Bar Display issues
    - Occasionally the HP bar will display a different colour than what it is supposed to be, or it will split colours about halfway. This issue is unresolved at the moment.
- Unbalanced Pokemon stats

    - The player's team of Pokemon had stats that made the game unbalanced; the player's Pokemon had too much of an advantage on the opponents. We tweaked the numbers in the game and balanced it out so that the player would have more of a challenge playing the game.
- File size
    - Due to the opening sequence, and the transition sequences in our game, the entire file of the game ended up being around 200+ megabytes at first. We reduced the file size by lowering the frames per second of the opening animation, and by converting all low colour depth images to a .gif file format.
    - We converted integer variables to short and byte variables in order to save space as well. Integer variables take up more space, as they record data that can go up to 2^32. Bytes record numbers that go up to 2^8, and shorts record numbers that can go up to 2^16. This saves a lot of memory, as the program doesn't have to allocate space for a number up to 2^32.
- Graphics
    - In our game, if your Pokemon runs out of PP for a move or faints, the box in which the game displays this information turns red. We had an issue where even after beating the opponent, the box would remain red even though the Pokemon had been restored completely.

**Improvements:**

As with everything, our game can use many improvements. Although most of them are not game-breaking, they can further enrich the player's experience and add another level of strategy to the game. For example, we could include the use of items in our game. In the original pokemon games, the player usually faces the Pokemon League prepared with many hp and pp restoring items, as well as revives. Another thing we can have to improve our game is to include the use of holdable items, as well as letting the player choose which item they want on their team before facing the elite four. These improvements would add another layer of choice into the game and increase its strategic value.

Aside from implementing new features into the game, we could also improve by continually fixing bugs that come up as we play the game (As seen through all the patches and versions). One long standing bug is when the opponent pokemon attacks, its hp bar sometimes changes colour and sometimes glitches out a bit. We already have a good idea of the source of this bug, but have yet to find a suitable method to eliminate it.

**Personal Remarks:**

David:

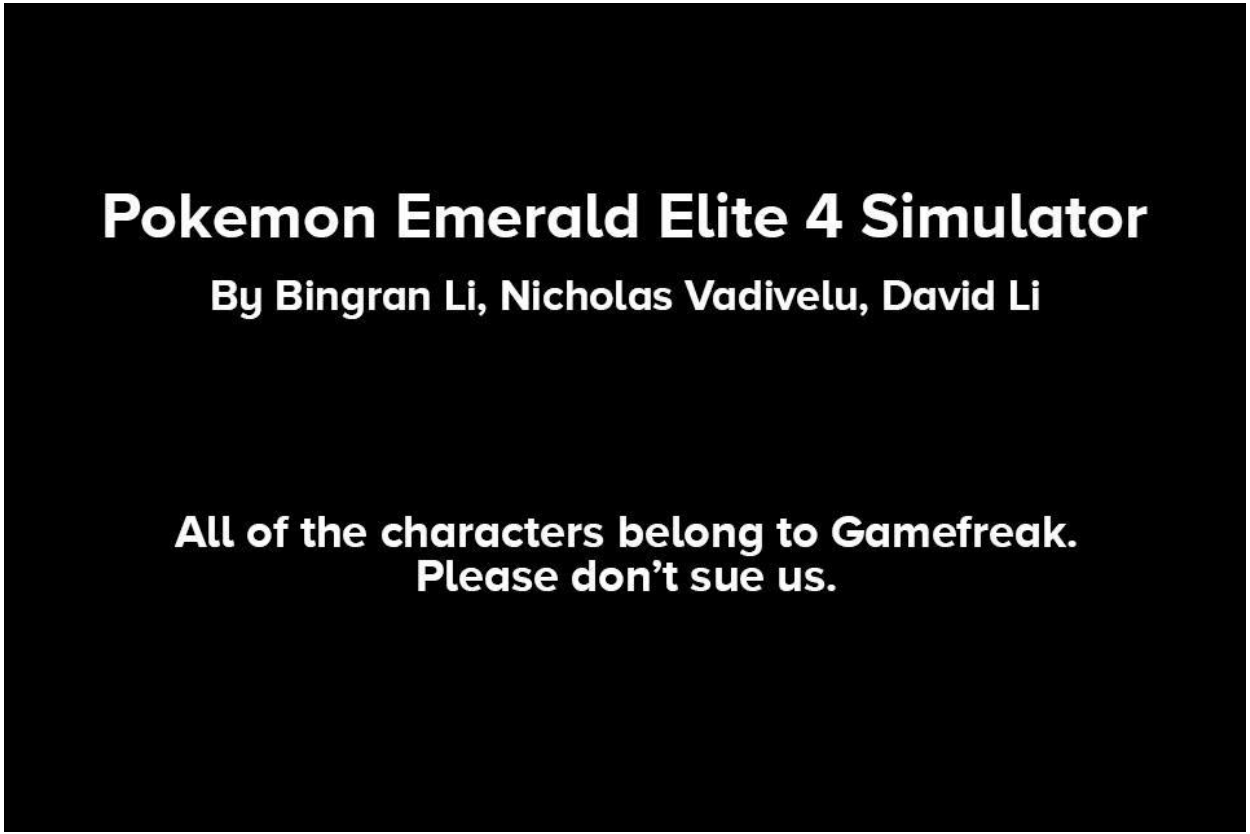
This assignment was very enjoyable for me, as I learned many new things while working on it. The whole idea of making a Pokemon game in Java was very interesting to me, as Pokemon was one of the games I grew up playing as a child. For me, the most enjoyable part of the assignment was playing the game to test for errors. We got to see how our work was coming along, and it was great to look back on it

and realize that we created this ourselves out of pretty much nothing. What I disliked about the assignment was going back and checking every line of code for one error that was messing the entire program up. It was very tedious and time consuming, and caused a lot of frustration in our group.

# ICS Summative Storyboard

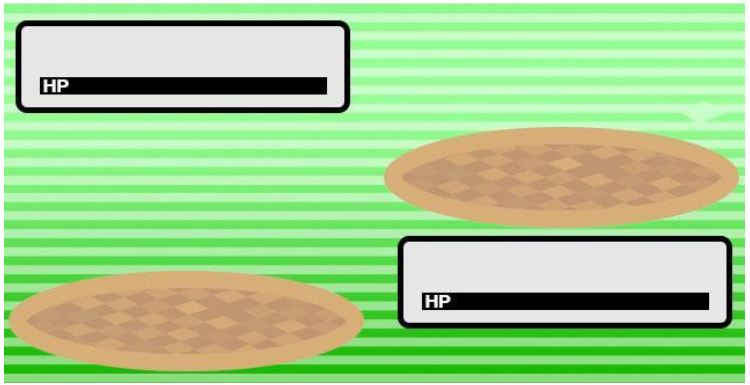
David, Bingran, Nicholas

# Credits

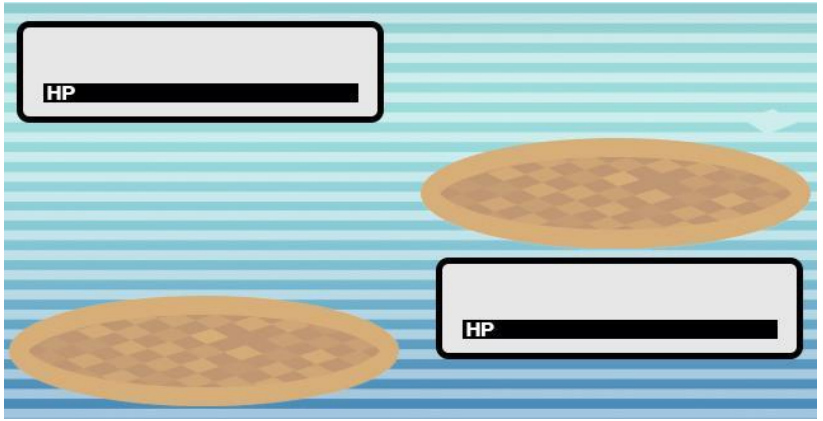

Pokemon Emerald Elite 4 Simulator
By Bingran Li, Nicholas Vadivelu, David Li
All of the characters belong to Gamefreak.
Please don't sue us.

# Opening Sequence

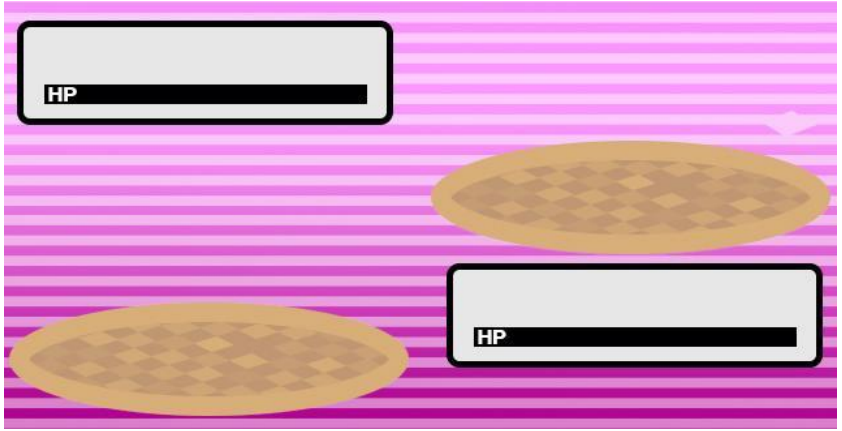
# Sidney Battle:



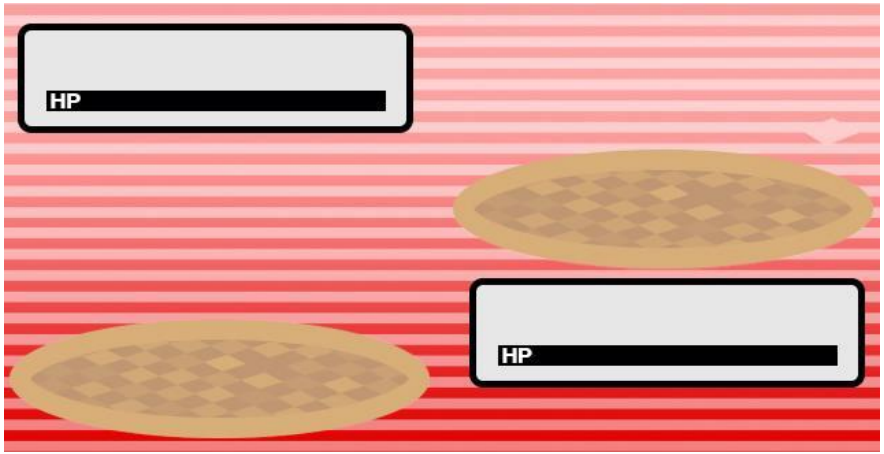
# Phoebe Battle:

# Glacia Battle:

# Drake Battle:

# Wallace Battle: