***Final Project: Pokenom***

In this program, we will redesign a game that has been a classic in many of our lives. For a large portion of our lives, the Nintendo Classic Pokemon has been created and recreated to fit the many changes and ideas that permeated through the times. The original style of the game that many have played throughout our childhood, however, has changed very little. From the regions of Kanto to the Islands of Alola, the core idea behind the Pokemon has always remained the same, keeping players engrossed throughout many years. The idea, as simple as it was, was able to keep players glued to their small screens. The role-playing-game, with quests and all sorts of interactions, remained the same for a very long time, until recently, as graphics improved.

In our project, we plan on recreating this coveted childhood game, but with its own little quirks. The original games’ simple design made it rather easy to recreate. However, as the gameplay was almost exactly the same, only spruced up with new characters and pokemon, we decided to add other function and restrictions to allow more interactive and thoughtful gameplay. It was only until the last two generations of games that the games began to include better graphics and a more realistic setting. However, we’ve decided to go back to basics, and begin again with the simple two dimensional maps and interactions with objects.

Our project, appropriated named Pokenom, combines the style and characters of pokemon with a unsatiated need to consume food. The gameplay is simple, similar to pokemon in several ways, yet also unique in ways that this game much more interesting that a simple Pokemon game.

The Gameplay is deceptively simple, as the player plays on a grass covered map, moving the pokemon, Munchlax, around, and collect berries. The player gains levels for collecting berries. However, each berry has a 20% chance to be a monster. The game continues this way until either the player dies in a battle or the player decides to stop the game. The main point of the game is to hit the highest level possible.

The player, the Munchlax, will run around collecting fruit while fighting enemies, gaining experience for each berry eaten or monster destroyed. As the Player gains experience, they gain levels. Eventually, as the levels get higher and higher, the game will come to a conclusion. The Player, Munchlax, system will keep track of several variables, such as health, level, and experience. If the player’s health hits zero, then the player dies, and will need to restart the game.

Experience is also another section that needs to be focused on. After each battle, if the Munchlax wins,he gains a set amount of experience. Eating berries will also provide a set amount of experience. After hitting a certain amount experience, the Munchlax will level up, gaining more health, and will learn a new move at certain intervals. As the Munchlax levels, the experience bar will get larger and larger, requiring more and tougher monsters to beat to fill up the bar.

Levels are a set of numbers that describe the Munchlax. With each level, his health will increase.

Moves are the set of abilities that Munchlax will be able to use during battle. Each one has a certain amount of power, and a certain amount of damage.The player begins with all the possible moves that Munchlax can use.

The Game is coded within Greenfoot, an interface that allows easy manipulation of GUI and background maps. The code is still in Java, just it outputs within the GreenFoot frame and workings. The interface is a full IDE which includes project management, auto-completion, syntax highlighting, and other tools common to most IDEs.

The gameplay relies on interactable objects and a map for it to work properly. The map and background, coded for in GreenFoot, create the plane that the Munchlax can run around on, similar to gridworld. However, the map that is used for the game is open and allows for freerunning. The Player is not limited to single grid movement, and can freely move across the map as they wish. The map prevents the objects from exiting the playing field.

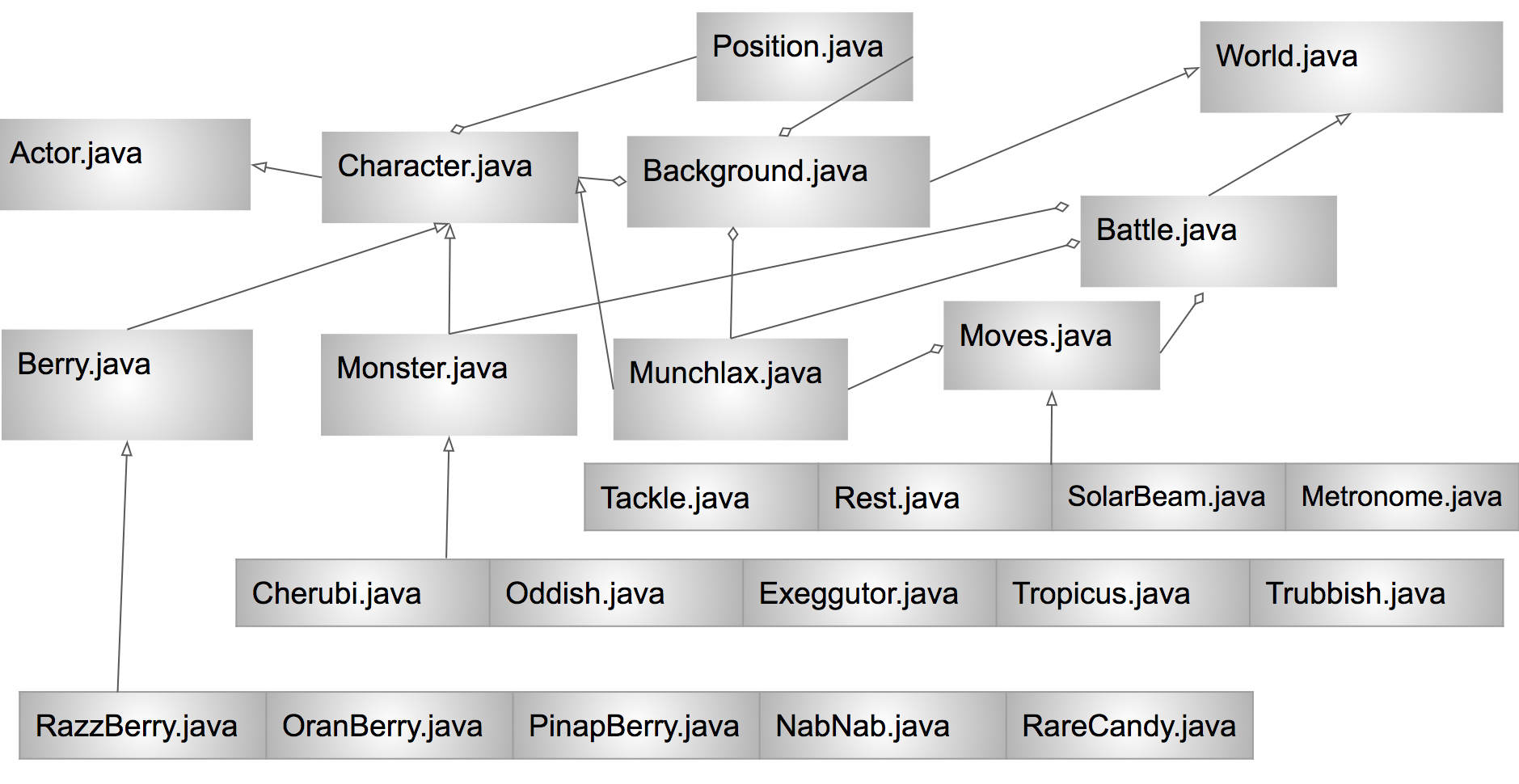
Interactable objects are the berries/monsters. When the player’s character object hits the berry or monster, it will trigger a specific action. If the interaction was with a berry, the experience marker will increase, and with it, a level, if so required. However, if it with a monster, the game will cut to another background, and the player will engage in battle with that monster.

The Battling code is a separate section from the rest of the program. Once you engage in battle, the background is set and the game batting commences. The Player will have choices to use different attacks during the course of the battle. These enemies are disguised as fruits, so the player has no idea if the fruit he is interacting with is a fruit or an enemy. After hitting an enemy, the player must fight until one faints. If the player wins, he continues the game, and gains a substantial amount of experience. If he loses, it's a game over.

Structural Design- We used a variety of data structures to hold the main objects that are implemented within the game.

1. Queue”To hold data of each object, we utilized several Arraylists. One array holds the instances of berries, one holds the instances of monsters,and the las holds the positions of the characters. This allows easy comparison and access of the objects and their positions for interactions. The way it works is: Every time the act is called, it creates a monster object and a berry object. After that, every 250 times the act method is called, one of the monster or berries will be removed from the queue and displayed on the screen.

Object Oriented Design



Detailed Design.

The detailed specifications for the game have been generated from Javadoc comments. They will be included in the projection submission, along with status reports, project plans, and Project Specifications.

Testing

There needs to be large and extensive testing required to ensure a quality product. It cannot simply be a Junit test that makes sure the methods works as it should. We as a group must also continually beta test the game and pull every single runtime error out of the program. As with every other game, this is a vital step in ensuring that the game runs as it should. Characters should be tested separately from the Grid.