**Pokemon Summary**

When initially brainstorming our project idea, our group split up into four separate sub-groups that would divide the task of writing a Pokemon game. These sub-groups were the player class, battle loop class, pokemon class, and the overworld class. As the overworld class, we were responsible for making what the player would actually interact with and run through (essentially the environment). At first, the idea was proposed to make ASCII graphics for the game so that it would be possible for the game to feel a bit more like the actual Pokemon game. This turned out to be too much of a task for our group. We decided it would be best to implement graphics in the next part of the development of the game converting our code to a game engine called RPG Maker.

As of now, the player can interact with the environment through text only; essentially making Pokemon a text-adventure that runs from the starting area “Pallet Town” to the first gym in the game that is in “Viridian City”. We cut some content for the player to be able to challenge the first gym, otherwise, the game would be significantly outside the scope of a class project.

The pokemonClass has been added and implemented into the various other classes. With the pokemonClass pokemon battles have been implemented, these battles take place in the various areas created by the Overworld class. The Overwold consists of two towns with separate houses and buildings, an intro area and a route with grass area where the player can battle the 13 wild Pokemon and allows back and forth travel between the towns. Each town has it in habitents with dialog. While in a battle the player has the options to attack, use an item, run away and to switch Pokemon which is unavailable in this version of the game. There are 13 overall pokemon each with their own stats, types, and movesets to battle with along with a gym leader battle.

**Under Development**

At the moment we are currently looking into making these features of the next version of the game. We definitely want it in the final version of the GUI implementation, at this time we wanted to limit the scope of the game and focus on the core of the game which is battling pokemon and gym leaders.

* Catch wild pokemon
* Run from Wild battles
* Pokemon Center Heals
* Pokemart
* Quests
* Pokemon Switching

**Overworld Summary**

When initially brainstorming our project idea, our group split up into four separate sub-groups that would divide the task of writing a Pokemon game. These sub-groups were the player class, battle loop class, pokemon class, and the overworld class. As the overworld class, we were responsible for making what the player would actually interact with and run through (essentially the environment). At first, the idea was proposed to make ASCII graphics for the game so that it would be possible for the game to feel a bit more like the actual Pokemon game. This turned out to be too much of a task for our group as soon, we would be converting our code to a game engine called RPG Maker.

As of now, the player can interact with the environment through text only; essentially making Pokemon a text-adventure that runs from the starting area “Pallet Town” to the first gym in the game that is in “Viridian City”. We cut some content for the player to be able to challenge the first gym, otherwise, the game would be significantly outside the scope of a class project.

**Overworld - Zachary**

Our group took on the task of trying to recreate the environments of the game, mainly by using the script of the game we found online. At first, we split up into groups for the first town, Pallet Town, in which I was in charge of creating the tutorial scenario “Professor Oak’s Laboratory”. I was quite apprehensive at first about the idea of combining code with other people just because I had no idea how it would really work considering there are so many options on how to tackle just one problem. The more I became familiar with my teammate Joshua the easier it was to see how we would be able to work together to make a product we are both proud of. It still needs a lot of work and it was not what I imagined we would make when I first signed up for the RPG group but I certainly had a lot of fun making this section of the game and I look forward to working in RPG Maker to bring the game fully to life.

**Overworld - Joshua**

The overworld team set out to create the environment of Pokemon. The original game spans 30+ locations with different environment biomes, from caves to oceans, forests, and grass fields. Our initial challenge was to adjust the scope of the overworld we were to create to match the timeline and deadline we had available. With that, we decided on the first two towns and the route between them. As a group, we sectioned off parts of the introduction town in town into three parts. It was the player house, The town, and Professor Oak's lab. I worked on the player house and Zach worked on the laboratory. We had another member in charge of the town layout, but after their departure, Zach completed their part. From there we combined our code and we were surprised that our programming methods were very similar. Combining our code so easily allowed us to break up parts of the code and work on them separately while taking the best pieces from each.

I believe we are able to work so well together because each of our code was clear and legible with plenty of comments. It was also very easy to communicate, we took our parts seriously and were slightly competitive. Which I think led to us working harder on our sections.

**Pokemon Class Summary - Andrew & Mario**

The Pokemon Class Team consists of both Andrew (me) and Mario who worked closely together over the course of the last several weeks to develop the Pokemon Class for the RPG Group Project. First, we started off by mapping out what an ideal C++ Class would like for a game of this category by designing a flowchart. From there, we began to create the attributes that would be allocated to every Pokemon such as name, type, and various combat stats. Shortly thereafter we made constructors for every Pokemon that was to be included in the game, a total of 13 Pokemon with 18 attributes each. Functions were created towards the end of the timeline, most importantly the Damage Multiplier Function which calculates the Damage dealt and received by Pokemon in battles. Initially, Mario and I ran into some difficulties creating the Damage Multiplier Function, but with the help of our other teammates we were able to get a better idea of how to create it.

Progress throughout the Pokemon Class portion of the project was not linear, but rather a zig zag. Sometimes when we met up we were able to collaborate effectively on the code, coding up to 100+ lines a session. Other times, we were struck with the misfortune of errors in the program, writer's block, and problems with trying to set up a platform for real time collaboration on the code i.e. fixing technical difficulties with Discord. In an attempt to give credit where credit is due, Mario and I would like to thank Thomas for his support in the overall design of the Pokemon Class. Thomas helped put the final touches on the Class, adding upon the Damage Multiplier function, creating the Move Struct, as well as including several other important functions. All in all, this project was an amazing learning experience for each of us and we couldn’t have asked for a better team.

**Player Class Summary - Junwei Yan**

The member of the player class group is Junwei Yan (me). Originally, there were two people in this group, but we were very confused about how to start and how to make it. At the beginning, we almost didn't write code, only looked up information on the Internet. When I was the only one in the back group, I started to ask Thomas and Zachary for help. The main function of the player class is actually a database, which is used to store data and call data. After I asked for the help of the team members, I began to review the use of class and vector, and started to complete the code at the same time. I have created two classes, one is item class, one is player class, and the other is Pokemon class. In player class, I have realized the functions of using items, adding Pokemon, switching Pokemon in battle, and switching Pokemon in a backpack to Pokemon stored in the computer. I think the whole process is very painful. I feel anxious about this part of the code, and I think I will delay it. In fact, I did drag the team down. But the team members didn't dislike me and gave me a lot of help. Thank you very much.

**Combat Event / Battle loop Class Summary - Thomas Gatewood**

I was the only one in this group, and while the amount of code was large, I was experienced in creating these turn based loops, and thus decided it would be better for me to take the entirety of this part. The battle system is the core gameplay, and is what pieces together many things in the project, so my initial drafts were merely small samples with placeholder code, which had to be redone later to fit everything from the pokemon and player classes. The code was created somewhat linearly, and was done in the order of turn progression. Pointers pointing to the pokemon allow for any pokemon to be put there, and still refer to specific pieces of that pokemon. A lot of the code is very similar, and I feel I should have made multiple functions within, such as making one for accuracy checks, one for the win/loss conditions, etc. Additionally, I could have more variables in the actual class description which would help with carrying these around the functions, such as the pointers. There’s plenty to improve and fix within this code, but it still turned out well, and will be improved when we move onto the next part of the project. Along with the fixes above, I hope to add more to the combat, attempting to stay more true to the games, such as stat buffs and debuffs, and

**--Pseudocode--**

**Intro:**

//Programmer: Joshua Buckles, Zachary Romero

//Assignment: Pokemon Group Project

//Description: The purpose of this program is to create a Intro Class

//PreProcessor Directive

//This line of code causes the contents of another file to be

//inserted into the program

//string to store PlayerName

//string to store RivalName

//int to store intro

//create GameIntro Member Function

//create GetPlayer Member Function

//create GetRival Member Function

//create Playerroom Member Function

//create Playerhouse Member Function

//create PalletTown Member Function

//create ProfessorOak Member Function

//create DefaultPalletTown Member Function

**Playerroom Function:**

// Display the menu.32

// Validate the menu selection.

// use GaddisStartingOutWithC pg245

// code to be executed if n = 2;

// code to be executed if n = 3;

// code to be executed if n = 4

**Playerhouse function:**

// Display the menu.32

// Validate the menu selection.

// use GaddisStartingOutWithC pg245

// code to be executed if n = 2;

// code to be executed if n = 3;

**PalletTown function:**

//this begins when exiting the starting house

//blue's house

//this would be where the player would receive the map if they have their pokedex from prof. Oak

//inside professor oak's lab before going to grass

//talk to aides

//talk to rival

//book

//computer

//posters

//poke balls

//this is when professor oak's scenario would pop up

//return grass;

**ProfessorOak function:**

//RED is dragged back to Professor Oak's house to select their POKeMON

//this is where we'd put the name of the rival

//RED is at Professor Oak's house, this is the initial bit of dialogue before RED can select their POKeMON

//now RED can choose their POKeMON: charmander, bulbasaur, or squirtle

//this is where we describe and have the POKeMON's stats and description

//section for choosing one of the three pokemon

//the rival is going to be choosing the opposite type pokemon

//this is where the first battle will take place, for now i'll just have a dice roll i favor of the rival

//the player needs to roll a 75 or higher to win the roll

//the player can choose whatever they want to do past this point

**Route1**

//Programmer: Joshua Buckles, Zachary Romero

//Assignment: Pokemon Group Project

//Description: The purpose of this program is to create Route1 class

//PreProcessor Directive

//This line of code causes the contents of another file to be

//inserted into the program

//Create Route1

//tile variable to hold player movement in grass

//choice variable to hold player choice in menu switch case

//end variable to hold bool for do/while loop for menus

//Creates member function to the runs introduction to the game

//Creates member function that returns an int variable tile

//Creates member function that runs battle encounter

//Creates member function that runs DefaultPalletTown

/\* ROUTE1\_H \*/

**RandomEncounter**

//Random Encounter function that runs a battle simulation as a place holder for battle class

//Create int randEncounter variable

//function to seed rand with system time

//each tile will have a 20% chance to spawn a random encounter

random encounter //This is where the random encounter will take place.

**Route1Intro**

//Function for to introduce players to grass and its functions

**Route1Defualt**

//Function to run after intro

//Do while to run menu for Grass

//Player movement menu that stores an int return value

//needs player class in the parenthesis

//Runs above menu until parameters are satisfied

//Returns Tile variable to main

**DefaultPalletTown**

//runs Default version of Pallet town that is different from the Intro class

//Rival's house menu

//Rival's house

//inside professor Oak's lab before going to grass

//return to the grass

**Default Towns Class:**

**ViridianCity Function:**

//set up bool to run Do while loop for Viridian City Menu

//talk to pokemon trainer

//talk to old man

//enter the pokecenter

//enter the pokemart

//enter the gym to challenge brock

//leave to the grass

**PokemonCenter Function:**

//talk to customer

//talk to receptionist

//this is where some heal pokemon function is called

//leave outside to viridian city

**PokemonMart Function:**

//bool variable set as a parameter for Do while loop to run menu

//Ends do while loop by switching bool variable, runs Viridian City Function

//Check bool variable to run loop, iv bool is true, loop breaks and player enters another function

**PokemonGym Function:**

//Pokemon Gym Function

//set up Do while loop to run menu

//Check loop breaks and player enters another function

**BrockGym Function:**

//Gym leader battle simulation

//function to seed rand with system time

//Create int randEncounter variable

//Wins battle against Gym leader and ends game.

//Ends the game

**EndGame function:**

//Ends the game

**PokemonClass:**

//Programmer: Mario Cuaya, Andrew Spurling, and Thomas Gatewood

//Assignment: Pokemon Group Project

//Description: The purpose of this program is to create a PokemonClass

//string variable to hold Pokemon Name

//string variables to hold Pokemon Types

//int values that hold Pokemon health stat, current Health, Base attack damage, base speed, base special attack damage, base defense, and pokemon number ID

//void function that displays the attack menu used in the Combat Class

//void function that displays pokemon information

//void function that heals pokemon using an item

//void function that checks whether a pokemon will receive extra damage based on what type it's going up against

//void function that displays the pokemon's health

//struct that defines a pokemon's moveset

**sruct move:**

//string variable that defines a pokemon's moveset name

//string variable that defines a pokemon's moveset type

//int variable that defines a pokemon's damage amount

//int variable that defines a pokemon's move's accuaray

//End of sruct

//move object that defines a pokemon's moveset

//move object that defines a pokemon's moveset

**PokemonClass default constructor:**

//PokemonClass default constructor that setups a PokemonClass object

//End of PokemonClass default constructor

**SetUp function:**

//Void function that setups the various Pokemon

//switch case loop

//if Pokemon number is 1 then create a Pikachu Pokemon Object

//Pokemon name

//Pokemon type

//Pokemon type two

//Pokemon health stat

//Current health is equal to the pokemon's health stat

//Pokemon's base attack damage

//Pokemon's base speed

//Pokemon's base defense

//Pokemon's base special attack damage

//Pokemon number in the pokedex

//Pokemon's first move name

//Type of attack the move is

//Accuracy of the move

//Damage of the move

//Setting up the pokemon's second move, which basically

//is the same process as the first move

//End of case 1

//if Pokemon number is 2 then create a Bulbasaur Pokemon Object

//Pokemon name

//Pokemon type

//Pokemon type two

//Pokemon health stat

//Current health is equal to the pokemon's health stat

//Pokemon's base attack damage

//Pokemon's base speed

//Pokemon's base defense

//Pokemon's base special attack damage

//Pokemon number in the pokedex

//Pokemon's first move name

//Type of attack the move is

//Accuracy of the move

//Damage of the move

//Setting up the pokemon's second move, which basically

//is the same process as the first move

//End of case 2

//if Pokemon number is 3 then create a Charmander Pokemon Object

//Pokemon name

//Pokemon type

//Pokemon type two

//Pokemon health stat

//Current health is equal to the pokemon's health stat

//Pokemon's base attack damage

//Pokemon's base speed

//Pokemon's base defense

//Pokemon's base special attack damage

//Pokemon number in the pokedex

//Pokemon's first move name

//Type of attack the move is

//Accuracy of the move

//Damage of the move

//Setting up the pokemon's second move, which basically

//is the same process as the first move

//End of case 3

//if Pokemon number is 4 then create a Charmander Pokemon Object

//Pokemon name

//Pokemon type

//Pokemon type two

//Pokemon health stat

//Current health is equal to the pokemon's health stat

//Pokemon's base attack damage

//Pokemon's base speed

//Pokemon's base defense

//Pokemon's base special attack damage

//Pokemon number in the pokedex

//Pokemon's first move name

//Type of attack the move is

//Accuracy of the move

//Damage of the move

//Setting up the pokemon's second move, which basically

//is the same process as the first move

//End of case 4

//This process repeats for 9 more cases and does the exact same thing, which is setting up the moves and stats for each pokemon

**DmgPoke function:**

//Void function that checks effectiveness against all types of pokemon

//int variable that acts as a flag to check for move effectiveness

//this if statement checks for move effectiveness

//inside of the if statement there are three other if statements that check for the pokemon’s effectiveness against its types that it’s strong, weak, and neutral against

//if the move is only half effective against a certain type then the pokemon’s damage gets a 1/2x buff

//else if it is super effective against a certain type then the pokemon’s damage gets a 2x buff

//else if it is not effective against any type then it gets no damage buff

//This process is repeated for all types and is the same process

//After all the statements have been checked then the damage variable which contains the damage buff is changed based on the the pokemon’s defense

//if damage comes out to being less than 0 then the damage is set to 1

//Checks if the attack set the pokemon's health to zero

//Else subtract the amount the attack did to from the pokemon's current health

//Checks whether an attack was very affective based on the effectiveness variable

//set up in the begin, which was added throughout the if, else if, and else statements

//If effective then print it was effective, else print it wasn't effective

//Displays the amount of damage the attack did after all the checks

//End of DmgPoke function

**pokemonInfo function:**

//void function that displays pokemon information

**HealPoke function:**

//void function that heals the pokemon

//if the current health plus the amount heal is greater than

//the health stat of the poke then set the current health to

//the pokemon's health stat

//Else heal the pokemon

**ShowHP function:**

//void function that displays the pokemon's current health

**attackMenu function:**

//void function that displays the attack menu for pokemon

**CombatEvent Class:**

**Combat function**

//bool returns whether the battle was won (Running and catching counts as winning), or lost(all pokemon have fainted AKA currentHealth == 0)

//Pointer is used so that any pokemon can be referenced

//Set up the pokemon to the stats as indicated by their pokemonNum which is stored in the array

// PlayerTurn is very important, as it allows the enemy to attack if the player uses an item.

// yourpoke and theirpoke refers to the next spot on the array of pokemon

// Entire Combat loop

// Player's turn: important for moving back in the menus

//every setting of choice makes sure that the player's previous choices do not affect future ones

//cout the HP of both team's active pokemon

//non int inputs are filtered and fixed through the !cin, cin.clear(), and empty while loop

//Correct choice leads here where it branches off, still can return to loop above

//show attacks when the attack option is chosen

//non int inputs are filtered and fixed through the !cin, cin.clear(), and empty while loop

//Code for attacks is repeated multiple times, for the player, and enemy in cases where they use different attacks, and different turn orders

//Put simply, player/enemy attacks first, checks accuracy, does damage, then repeats with whoever didn't attack

//case 1 is attack one

//speed determines who goes first

// chance is the hit chance/crit chance

//if the chance is equal to the move accuracy, it crits, if it's less it hits, if it's greater, it misses

//win condition checks after damage if opponent has any pokemon left

//end of win condition

//the case where the enemy attacks first due to a higher speed stat, followed by you

// this is the hit chance/crit chance

//LoseCondition Checks if the player has pokemon left

//end of LoseCondition

// this is the hit chance/crit chance

//win condition checks if opponent has any pokemon left

//end of win condition

// this is the hit chance/crit chance

//LoseCondition Checks if the player has pokemon left, if not, they lose

//end of Lose Condition

// this is the hit chance/crit chance

//win condition checks if opponent has any pokemon left

//end of win condition

// player chooses attack two, nearly the same code as attack 1

//win condition checks if opponent has any pokemon left

//end of win condition

//once again, enemy attacks first followed by you

// this is the hit chance/crit chance

//LoseCondition Checks if the player has pokemon left

//LoseCondition

// this is the hit chance/crit chance

//win condition checks if opponent has any pokemon left

//win condition

// this is the hit chance/crit chance

//LoseCondition Checks if the player has pokemon left

//LoseCondition

// this is the hit chance/crit chance

//win condition checks if opponent has any pokemon left

//win condition

// ATTACK STUFF ENDS

//switching pokemon does not work here because the player has no other pokemon but the ability to do it would be here

// items

//potion heals the pokemon for 30hp

//CHECK FOR SPOT TO ADD IN PLAYER ARRAY THEN ADD THE DEX NUM

//playerArr[i] = EnemyPoke->pokemonNum;

// running works if the random number generated is greater than 20, essentially an 80% chance

//cannot run from trainer battles

//end combat if success

//enemy Turn now for when the player attacked first or did something other than attack

// this is the hit chance/crit chance

//LoseCondition Checks if the player has pokemon left

//LoseCondition

**Player Class** :

//Programmer: Junwei Yan

//Assignment: Pokemon Group Project

//Description: The purpose of this program is to create playerclass

**itemclass**

//create a object class

//Set the name and quantity of the item

**itemclass construct function**

//assigned name and num to the ItemNum and ItemName

**getNum function**

//Get the corresponding quantity of the item

**getName function**

//Get the corresponding name of the item

**showItem function**

//Display item name

//Display item quantity

**setNum function**

//Get the number of corresponding items(initial value)

**player class**

//create a vector for Backpack for storing items

//create a vector for This is related to Pokemon class

//create a vector for could save 11 pokemons

//create a vector for Pokemon used in battle

**getitem function**

//Get item information

**useItem function**

//Use items

for (each element in bag)

if the specific item is found

return the quantity of that item

if quantity is invalid //not integer, less than 1

return false;

else

substract quantity of the item

**showItemInfo function**

//Display item information

for(each element in bag)

show that element

**addpokemonToCarry function**

//Add Pokemon that needs to be around

if (pokemon bag is full) //already 6 elements in bag

print error message

else

add the new pokemon into the bag

**addpokemontoComp function**

//add pokemon to computer

**bagupdate function**

//update the data for the bag

**fightingPoke function**

//Choose the pokemon to fight (the first one by default)

if (there is no pokemon in fight)

add the first pokemon in bag to fight

delete the first element in vector

**getFightingpoke function**

//get the vector of fighting pokemon

**changepoketofight function**

//Pokemon can be replaced in battle

move the fighting pokemon to bag

then move the selected pokemon to fighting

**changePokemonfromComp function**

if(selected position of bag exceeds capacity of bag or selected position of computer exceeds capacity of computer)

return error

else

move the pokemon in the selected postion in the computer to specific position of bag