Programming Methodology
Project 1 Description

2024 Spring Semester

Machine
Intelligence &
Data science LAB

TA Information

- TAs
  - Dohoon Kim (dohoon.kim@snu.ac.kr)
  - Yongho Kim (peterkim98@snu.ac.kr)
  - Juhyeon Park (parkjh9229@naver.com)

- Post questions about the project on ETL

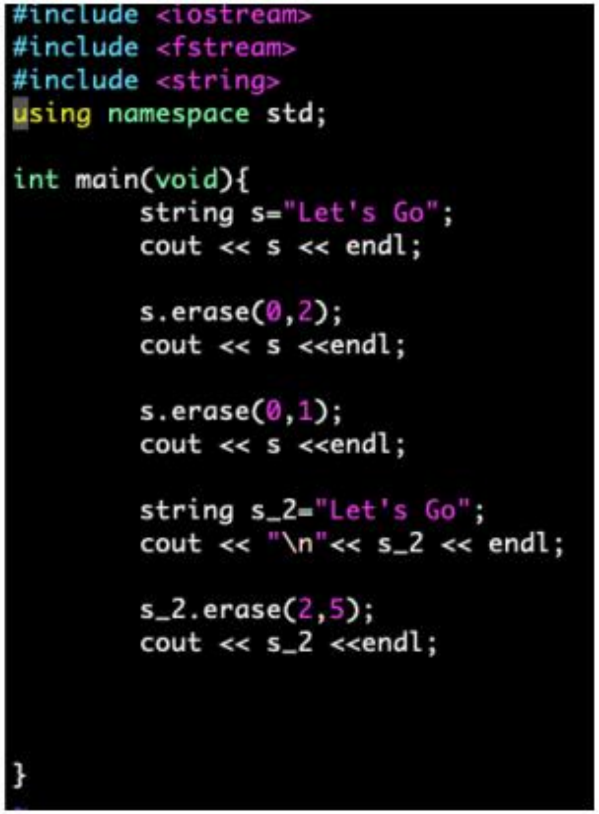
- DO NOT COPY OR CHEAT

Outline

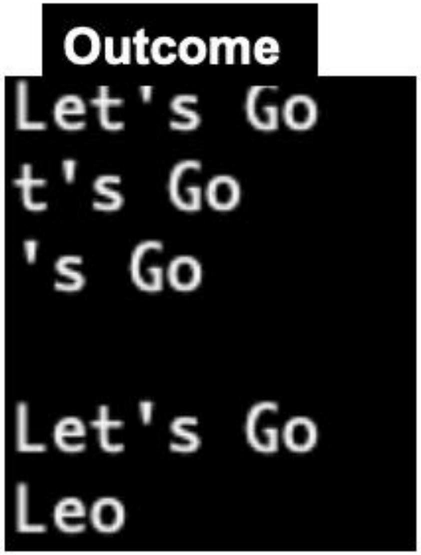
- Useful Standard Libraries
- Pokémon Battle
  - Demo & Rules
  - Code structure
- Submission and Grading

Useful Standard Libraries

- std::string
  - objects that represent sequences of characters.



```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main(void){
        string s="Let's Go";
        cout << s << endl;

        s.erase(0,2);
        cout << s <<endl;

        s.erase(0,1);
        cout << s <<endl;

        string s_2="Let's Go";
        cout << "\n"<< s_2 << endl;

        s_2.erase(2,5);
        cout << s_2 <<endl;



}
```

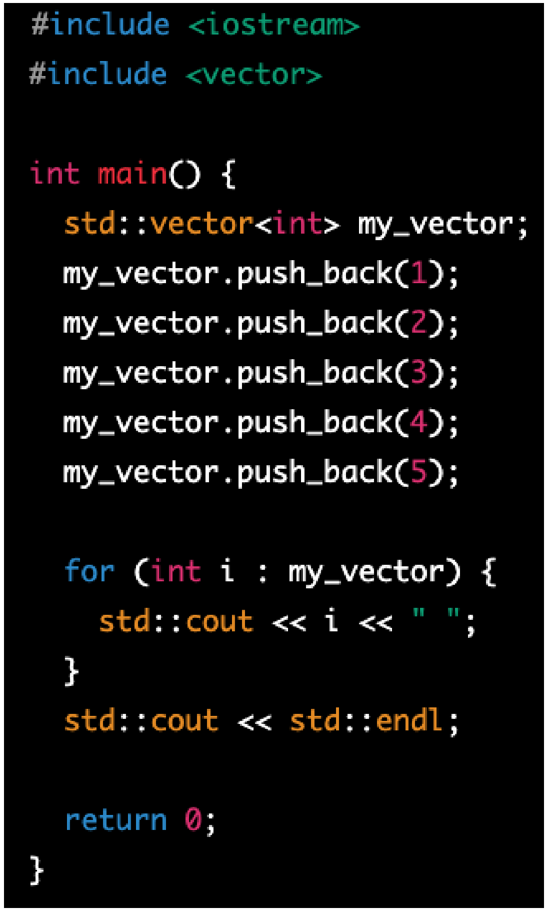


Outcome

```
Let's Go
t's Go
's Go

Let's Go
Leo
```

Useful Standard Libraries

- std::vector
  - A sequence container that encapsulates dynamic size arrays



```
#include <iostream>
#include <vector>

int main() {
  std::vector<int> my_vector;
  my_vector.push_back(1);
  my_vector.push_back(2);
  my_vector.push_back(3);
  my_vector.push_back(4);
  my_vector.push_back(5);

  for (int i : my_vector) {
    std::cout << i << " ";
  }
  std::cout << std::endl;

  return 0;
}
```

Project Overview

- Goal: Create a Pokémon battle simulator
  - 2-player game

- You will practice:
  - File I/O, terminal I/O
  - Creating/organizing class instances
  - Managing the flow of a big program
  - Very normal math algorithms

Demo

- Let's look at a demo of the project
- The demo is also available in Elice
- Your program's terminal output is not required to exactly match the demo
- You can reference the demo if you have questions about the game's rule

Game Flow

- Each player sends out a Pokémon to battle the other player’s Pokémon

- At each turn, each player’s Pokémon attacks the other player’s Pokémon or switches to another Pokémon

- Each player starts with 3 Pokémon

- The first player to defeat all of the other player’s Pokémon wins

Pokémon

- Each Pokémon has 6 stats:
    - HP: Health points, the Pokémon Faints if its HP falls to 0
    - Attack: Determines the damage of physical attacks
    - Defense: Determines the damage taken from physical attacks
    - Special Attack: Determines the damage of special attacks
    - Special Defense: Determines the damage taken from special attacks
    - Speed: Determines the order of attack during each turn

- Each Pokémon has up to 4 attacking moves

Game Flow

- At the beginning of each turn, the player can either:
  - Attack with the Pokémon using one of the attacking moves
  - Switch to another Pokémon (this consumes the turn)
  - End the game by typing -1

- If both Pokémon choose to attack, the Pokémon with a higher speed stat attacks first
  - You do not need to consider speed ties (when they have the same speed)

- If a Pokémon takes damage and faints, it cannot attack that turn, and the player must send out another Pokémon
  - Unlike switching, this does not consume the turn

Attacking Moves

- Attacking moves are split into 2 categories: physical or special
  - Physical moves use my (physical) attack and opponent's (physical) defense
  - Special moves use my special attack and opponent's special defense

- Attacking moves also belong to a certain type
  - Ex) "Fire blast" is a fire type move

- Moves have limited number of uses, called PP (power point)
  - Each move loses 1 PP after being used
  - Moves with 0 PP cannot be used
  - Pokémon without any available moves uses a special weak move named "struggle" with unlimited PP

Switching

- At each turn, the player can choose to switch their Pokémon
- The player cannot attack if they choose to switch
- Switching always happens first; if the opponent chooses to attack, the Pokémon that gets called takes the damage
- You can only switch to a Pokémon that is alive
- If both players switch, the faster Pokémon switches first

Damage Calculation

- Initial damage calculation
  - The damage taken from a physical attack is calculated as:

$$\frac{(\textit{Attack stat of attacking Pokémon}) \times (\textit{Power of attacking move})}{(\textit{Defense stat of defending Pokémon})}$$

  - The damage taken from a special attack is calculated as:

$$\frac{(\textit{Special attack stat of attacking Pokémon}) \times (\textit{Power of attacking move})}{(\textit{Special defense stat of defending Pokémon})}$$

Damage Multiplier: Type Effectiveness

- Each Pokémon and move belongs to a certain type
  - We will consider 4 types: fire, water, grass, normal

- The attack can to extra or less damage depending on the move’s (attacking) type and the receiving Pokémon’s (defending) type
  - Ex) A grass type Pokémon takes 2x damage from a fire type attack

	Defending Type				
Attacking Type		Normal	Fire	Water	Grass
	Normal	1x	1x	1x	1x
	Fire	1x	0.5x	0.5x	2x
	Water	1x	2x	0.5x	0.5x
	Grass	1x	0.5x	2x	0.5x

Damage Multiplier: STAB

- Same type attack bonus (STAB)
- If the type of the attacking Pokémon match the type of the attacking move, it will do 1.5x damage
  - Ex) If a fire type Pokémon uses a fire type attack, it will do 1.5x damage

- Final damage calculation
  - $\lfloor (Initial\ damage) \times (damage\ multipliers) \rfloor$
  - Damage multipliers include type effectiveness, STAB, and certain effects from held items

Held Items

- Each Pokémon can have up to one held item

- Each held item has a unique effect, like recovering HP or boosting the damage from an attack

- Each held item is either consumable or permanent
  - Consumable held items are consumed after being used
  - Permanent held items are not consumed after being used

Held Items

- Held items have two key components: condition and effect

- Condition refers to the condition which the item would activate
  - hp_below_threshold: when hp falls below a fraction of max HP
  - end_of_turn: at the end of each turn
  - move_type: when the Pokémon's attacking move is a certain type
  - move_category: when the Pokémon's attacking move is a certain category
  - damage_done: when the Pokémon deals damage

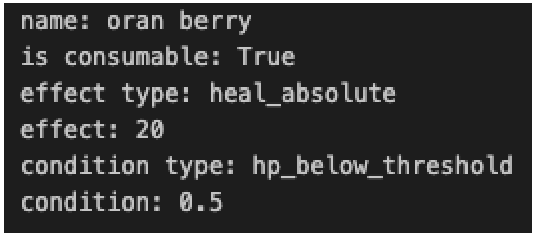
Held Items

- Held items have two key components: condition and effect

- Effect refers to the effect of the item
  - heal_absolute: heal the HP by a certain value
  - heal_relative: heal a fraction of the max HP
  - boost_move_power: boost the attacking move's power by a certain fraction
  - lifesteal: heal the HP by a fraction of the damage dealt
    - Only heal for the fraction of the actual damage dealt ex) if the defending Pokémon has 100 HP and the move would've dealt 200 damage, only heal for a fraction of 100 HP.

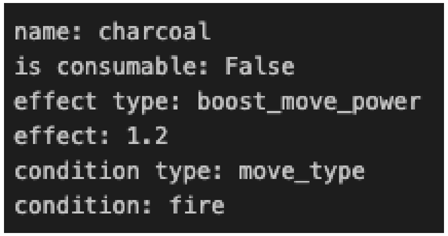
Held Items

- Examples
  - Heal 20 HP when HP is below 50%



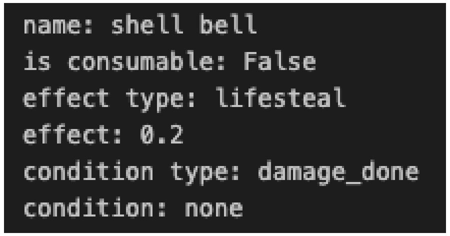
```
name: oran berry
is consumable: True
effect type: heal_absolute
effect: 20
condition type: hp_below_threshold
condition: 0.5
```

  - Boost the damage of fire type attacks by 20%



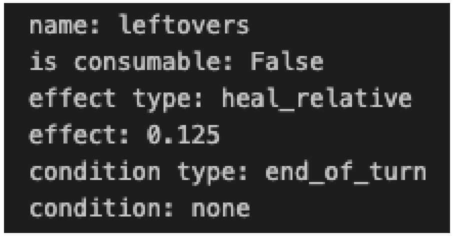
```
name: charcoal
is consumable: False
effect type: boost_move_power
effect: 1.2
condition type: move_type
condition: fire
```

  - Heal 20% of the damage dealt



```
name: shell bell
is consumable: False
effect type: lifesteal
effect: 0.2
condition type: damage_done
condition: none
```

  - Heal 12.5% of max HP at the end of each turn



```
name: leftovers
is consumable: False
effect type: heal_relative
effect: 0.125
condition type: end_of_turn
condition: none
```

Start of Game

- At the beginning, each player drafts 3 Pokémon from a given list
  - Player 1 picks their 1st Pokémon
  - Player 2 picks their 1st and 2nd Pokémon
  - Player 1 picks their 2nd and 3rd Pokémon
  - Player 2 picks their 3rd Pokémon

- 3 held items are drafted the same way

- Each player assigns a held item to their Pokémon

Struggle

- Pokémon without any available moves (all of the moves' PP=0) uses a special move called "struggle"
- Struggle is a normal type physical move with 30 power with 999 PP
- You have to hard-code struggle into the game (not from the moves list)

Outline

- Useful Standard Libraries
- Pokémon Battle
  - Demo & Rules
  - Code structure
- Submission and Grading

File I/O

- The list of available Pokémon, moves, and held items will be provided as a txt file

- You must write a code to process the information from the txt file to create appropriate class instances

- Formats for the txt files will be provided

Project Code Structure

- main.cpp
- game.cpp, .hpp
- player.cpp, .hpp
- pokemon.cpp, .hpp
- move.cpp, .hpp
- held_item.cpp, .hpp
- spreadCs.cpp, .hpp

Do not modify main.cpp

Project Code Structure

- You are free to modify any areas of the code outside of the "implement here" section
- You are free to define new functions as needed

Do not modify main.cpp!

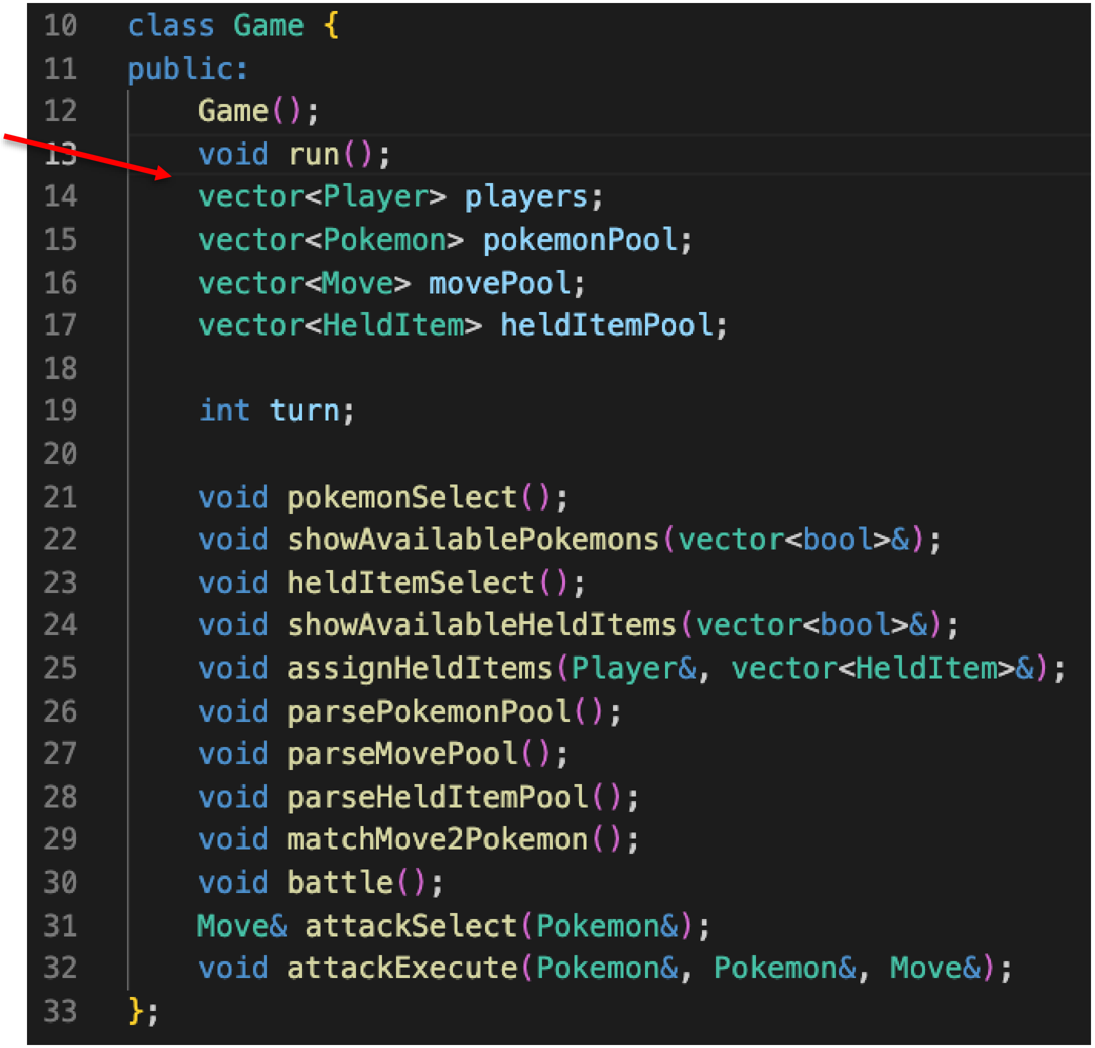
Project Code Structure

- game.hpp

'private:' is recommended here, but we omit it for evaluation

Member variables

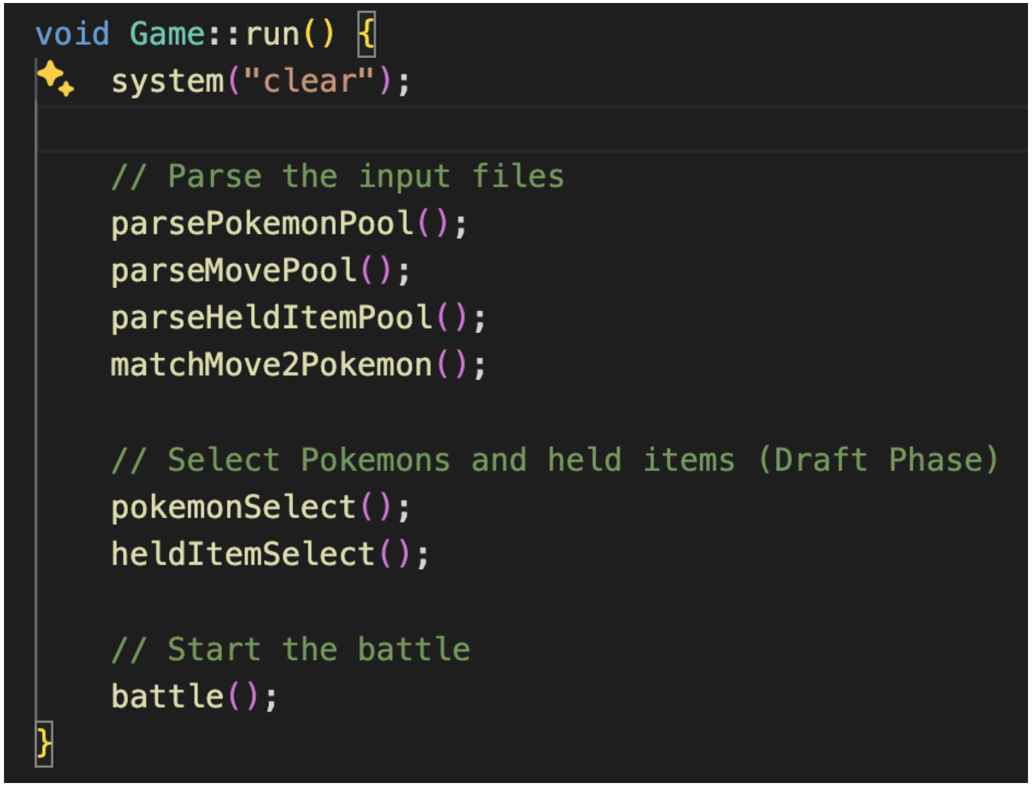
Member functions



```
class Game {
public:
    Game();
    void run();
    vector<Player> players;
    vector<Pokemon> pokemonPool;
    vector<Move> movePool;
    vector<HeldItem> heldItemPool;

    int turn;

    void pokemonSelect();
    void showAvailablePokemons(vector<bool>&);
    void heldItemSelect();
    void showAvailableHeldItems(vector<bool>&);
    void assignHeldItems(Player&, vector<HeldItem>&);
    void parsePokemonPool();
    void parseMovePool();
    void parseHeldItemPool();
    void matchMove2Pokemon();
    void battle();
    Move& attackSelect(Pokemon&);
    void attackExecute(Pokemon&, Pokemon&, Move&);
};
```

Project Code Structure

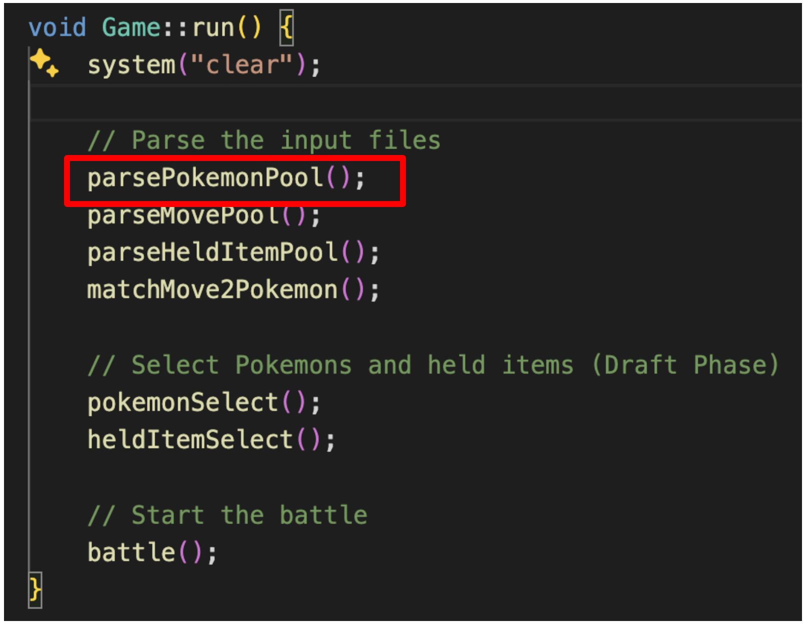
- game.cpp
  - Game::run() : overall flow of the game



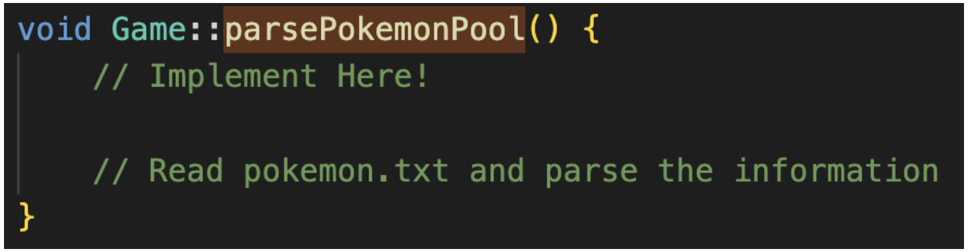
```cpp
void Game::run() {
    system("clear");

    // Parse the input files
    parsePokemonPool();
    parseMovePool();
    parseHeldItemPool();
    matchMove2Pokemon();

    // Select Pokemons and held items (Draft Phase)
    pokemonSelect();
    heldItemSelect();

    // Start the battle
    battle();
}
```

Project Code Structure

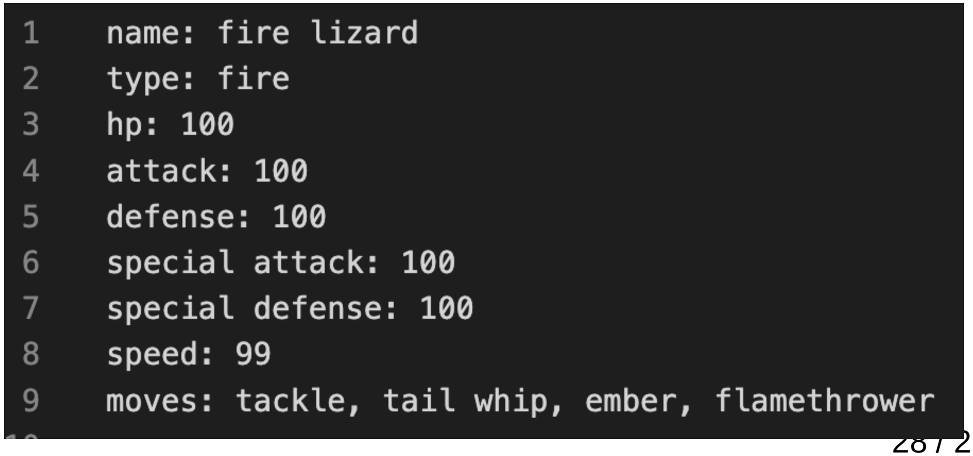
- game.cpp
  - Game::parsePokemonPool()
  - Store the parsed Pokemon information in the vector<Pokemon>pokemonPool



```cpp
void Game::run() {
    system("clear");

    // Parse the input files
    parsePokemonPool();
    parseMovePool();
    parseHeldItemPool();
    matchMove2Pokemon();

    // Select Pokemons and held items (Draft Phase)
    pokemonSelect();
    heldItemSelect();

    // Start the battle
    battle();
}
```



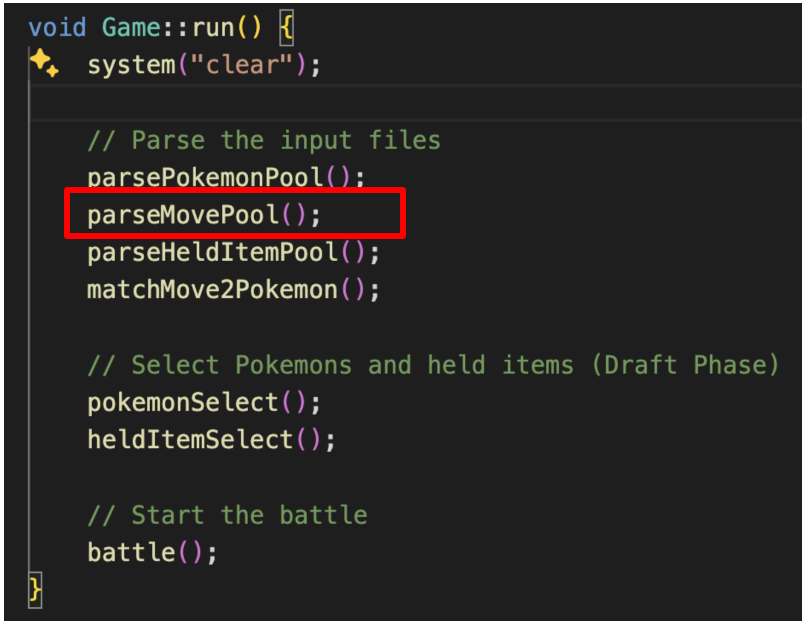
```cpp
void Game::parsePokemonPool() {
    // Implement Here!

    // Read pokemon.txt and parse the information
}
```



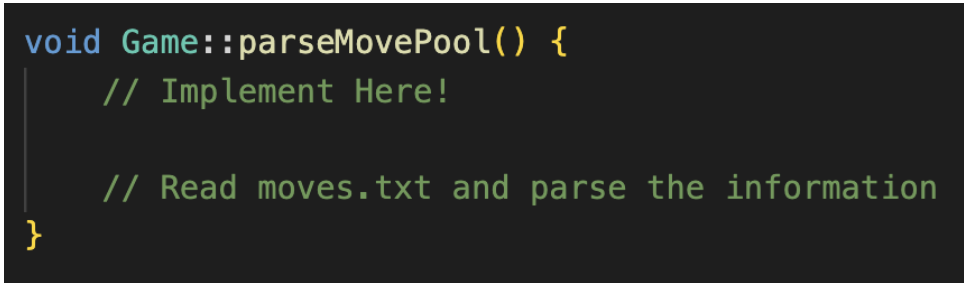
```
name: fire lizard
type: fire
hp: 100
attack: 100
defense: 100
special attack: 100
special defense: 100
speed: 99
moves: tackle, tail whip, ember, flamethrower
```

Project Code Structure

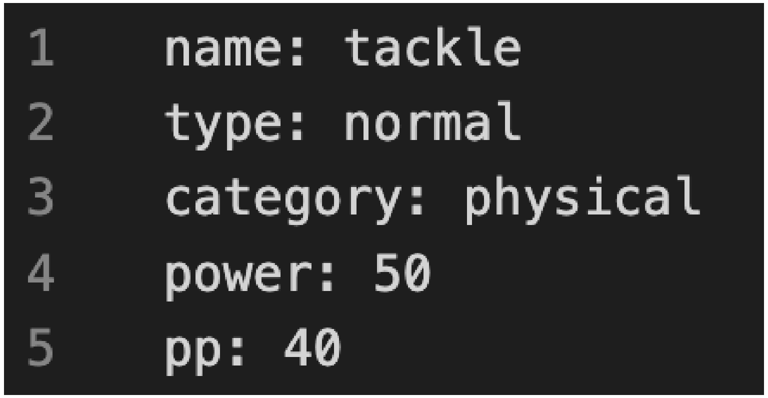
- game.cpp
  - Game::parseMovePool()
  - Store the parsed Move information in the vector<Move>movePool



```
void Game::run() {
    system("clear");

    // Parse the input files
    parsePokemonPool();
    parseMovePool();
    parseHeldItemPool();
    matchMove2Pokemon();

    // Select Pokemons and held items (Draft Phase)
    pokemonSelect();
    heldItemSelect();

    // Start the battle
    battle();
}
```



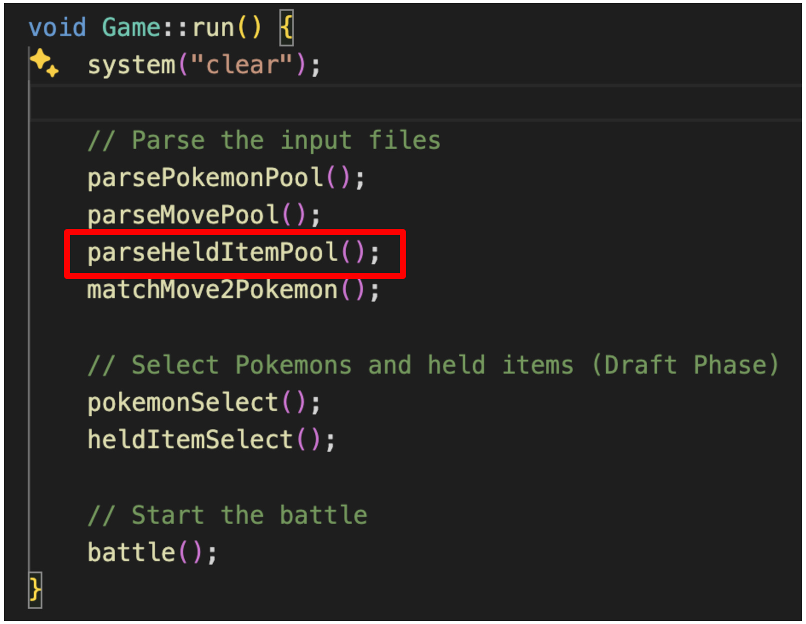
```
void Game::parseMovePool() {
    // Implement Here!

    // Read moves.txt and parse the information
}
```



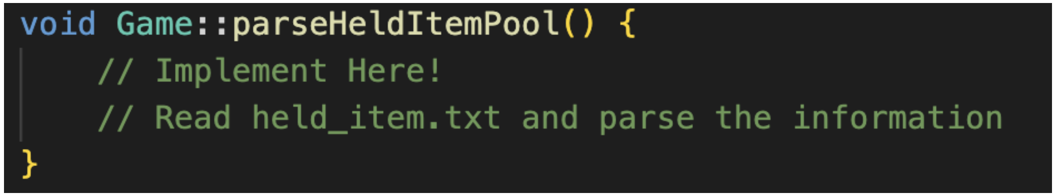
```
name: tackle
type: normal
category: physical
power: 50
pp: 40
```

Project Code Structure

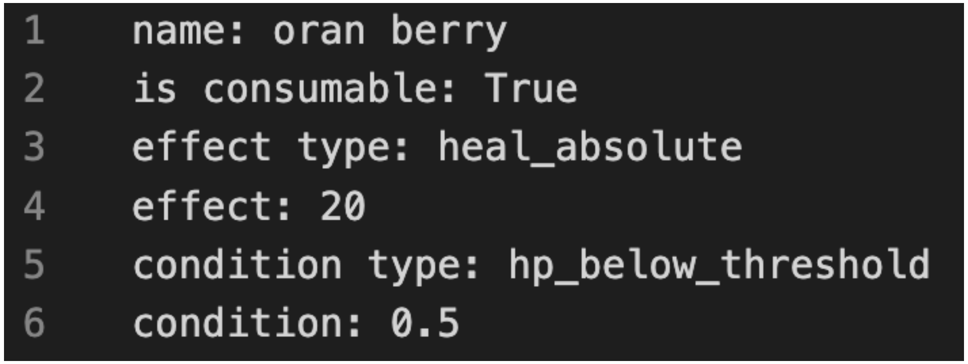
- game.cpp
  - Game::parseHeldItemPool()
  - Store the parsed HeldItem information in the vector<HeldItem>heldItemPool



```cpp
void Game::run() {
    system("clear");

    // Parse the input files
    parsePokemonPool();
    parseMovePool();
    parseHeldItemPool();
    matchMove2Pokemon();

    // Select Pokemons and held items (Draft Phase)
    pokemonSelect();
    heldItemSelect();

    // Start the battle
    battle();
}
```



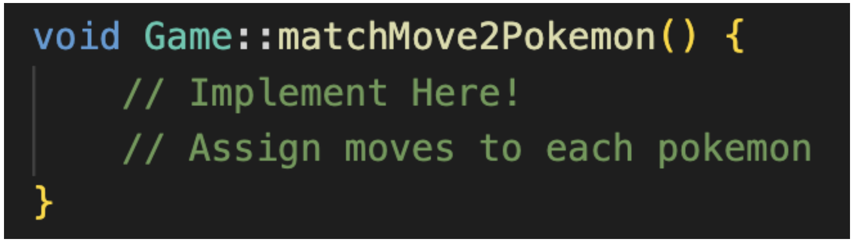
```cpp
void Game::parseHeldItemPool() {
    // Implement Here!
    // Read held_item.txt and parse the information
}
```



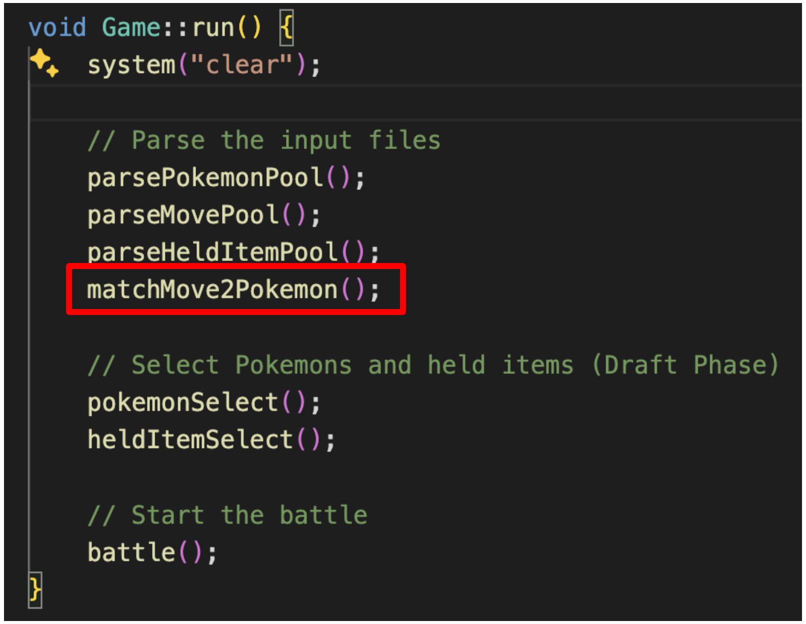
```
name: oran berry
is consumable: True
effect type: heal_absolute
effect: 20
condition type: hp_below_threshold
condition: 0.5
```

Project Code Structure

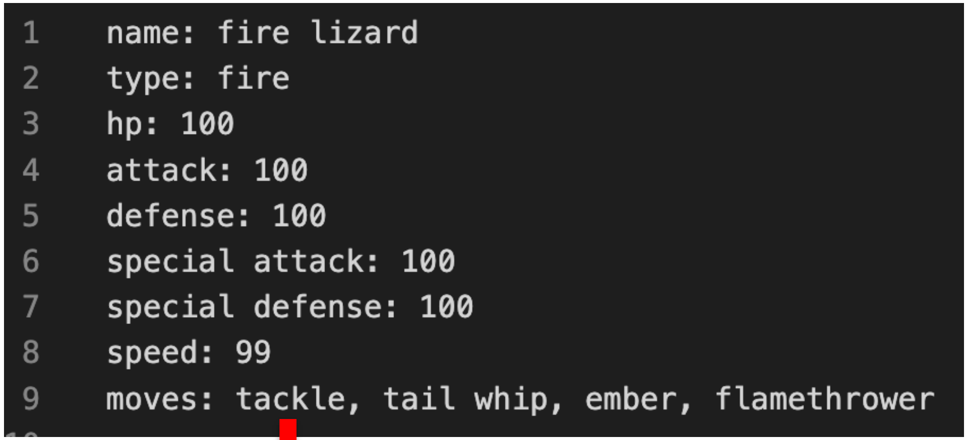
- game.cpp
  - Game::matchMove2Pokemon()
  - Assign moves' information to pokemon



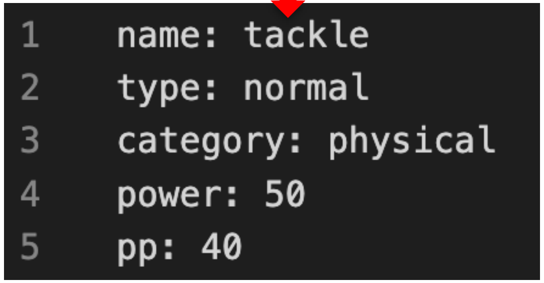
```
void Game::matchMove2Pokemon() {
    // Implement Here!
    // Assign moves to each pokemon
}
```



```
void Game::run() {
    system("clear");

    // Parse the input files
    parsePokemonPool();
    parseMovePool();
    parseHeldItemPool();
    matchMove2Pokemon();

    // Select Pokemons and held items (Draft Phase)
    pokemonSelect();
    heldItemSelect();

    // Start the battle
    battle();
}
```



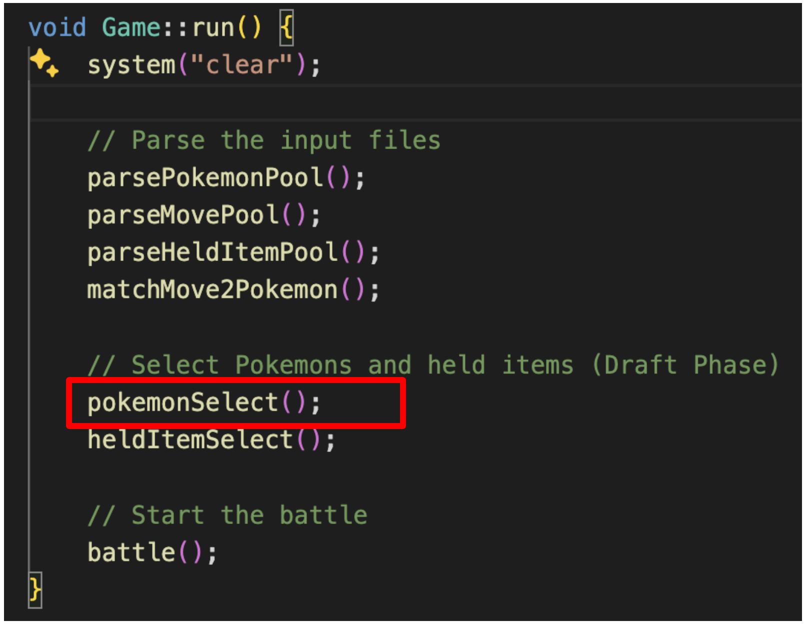
```
name: fire lizard
type: fire
hp: 100
attack: 100
defense: 100
special attack: 100
special defense: 100
speed: 99
moves: tackle, tail whip, ember, flamethrower
```



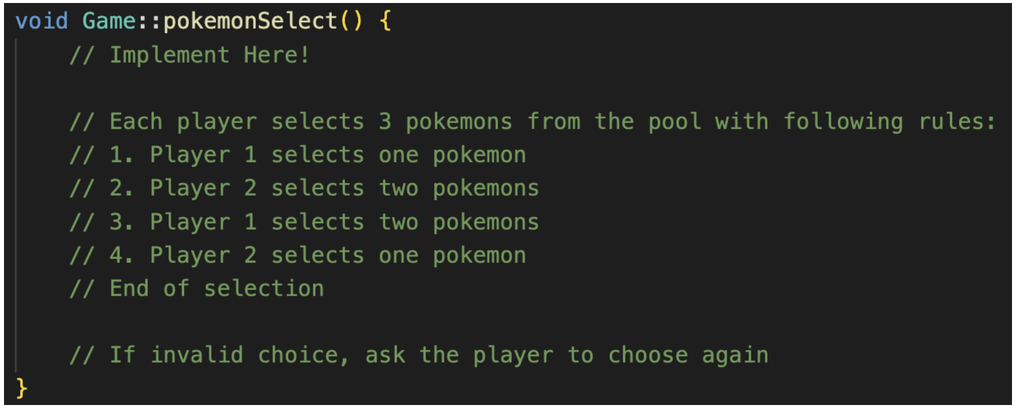
```
name: tackle
type: normal
category: physical
power: 50
pp: 40
```

Project Code Structure

- game.cpp
  - Game::pokemonSelect()
  - Each player selects 3 pokemons from the pool
  - Utilize `void showAvailablePokemons(vector<bool>&);`



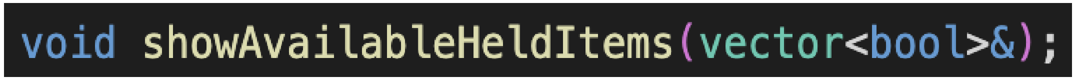
```
void Game::run() {
    system("clear");

    // Parse the input files
    parsePokemonPool();
    parseMovePool();
    parseHeldItemPool();
    matchMove2Pokemon();

    // Select Pokemons and held items (Draft Phase)
    pokemonSelect();
    heldItemSelect();

    // Start the battle
    battle();
}
```



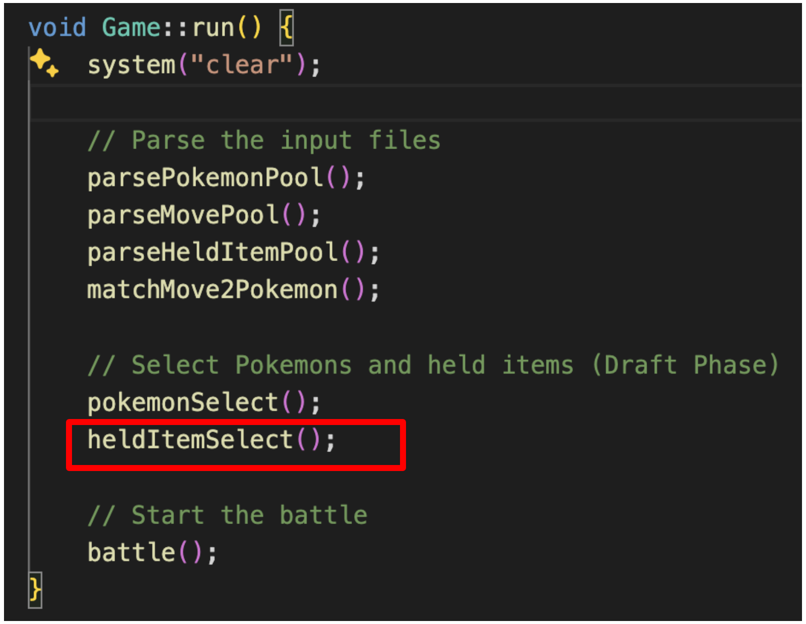
```
void Game::pokemonSelect() {
    // Implement Here!

    // Each player selects 3 pokemons from the pool with following rules:
    // 1. Player 1 selects one pokemon
    // 2. Player 2 selects two pokemons
    // 3. Player 1 selects two pokemons
    // 4. Player 2 selects one pokemon
    // End of selection

    // If invalid choice, ask the player to choose again
}
```

Project Code Structure

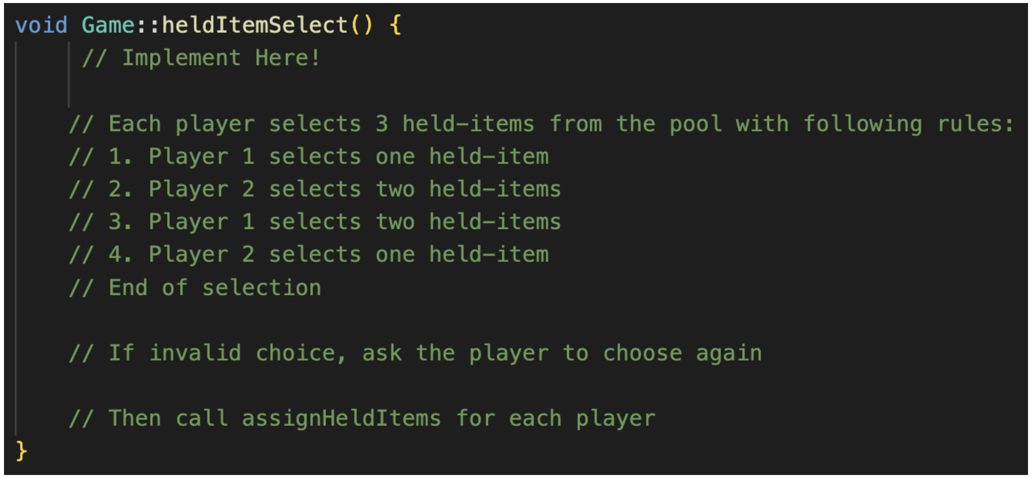
- game.cpp
  - Game::heldItemSelect()
  - Each player selects 3 held-items from the pool
  - Utilize



```
void showAvailableHeldItems(vector<bool>&);
```



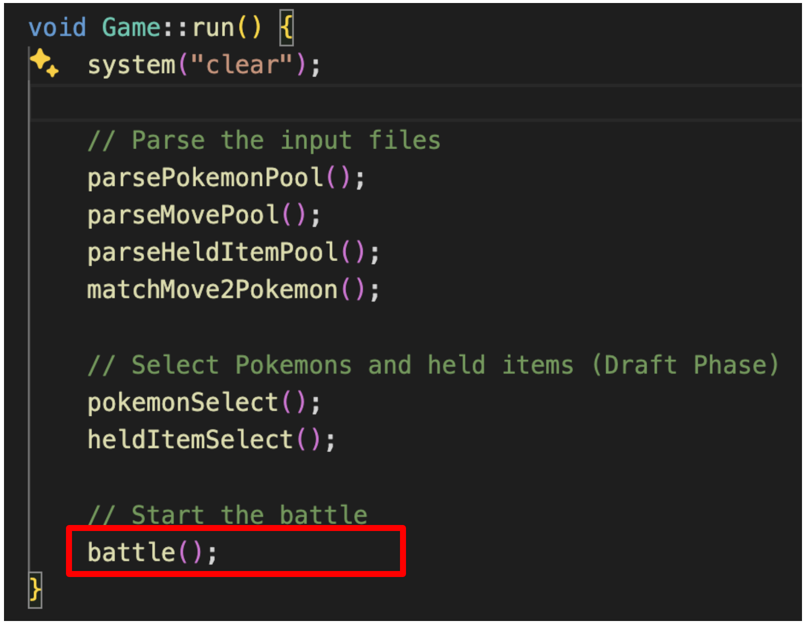
```
void Game::run() {
    system("clear");

    // Parse the input files
    parsePokemonPool();
    parseMovePool();
    parseHeldItemPool();
    matchMove2Pokemon();

    // Select Pokemons and held items (Draft Phase)
    pokemonSelect();
    heldItemSelect();

    // Start the battle
    battle();
}
```



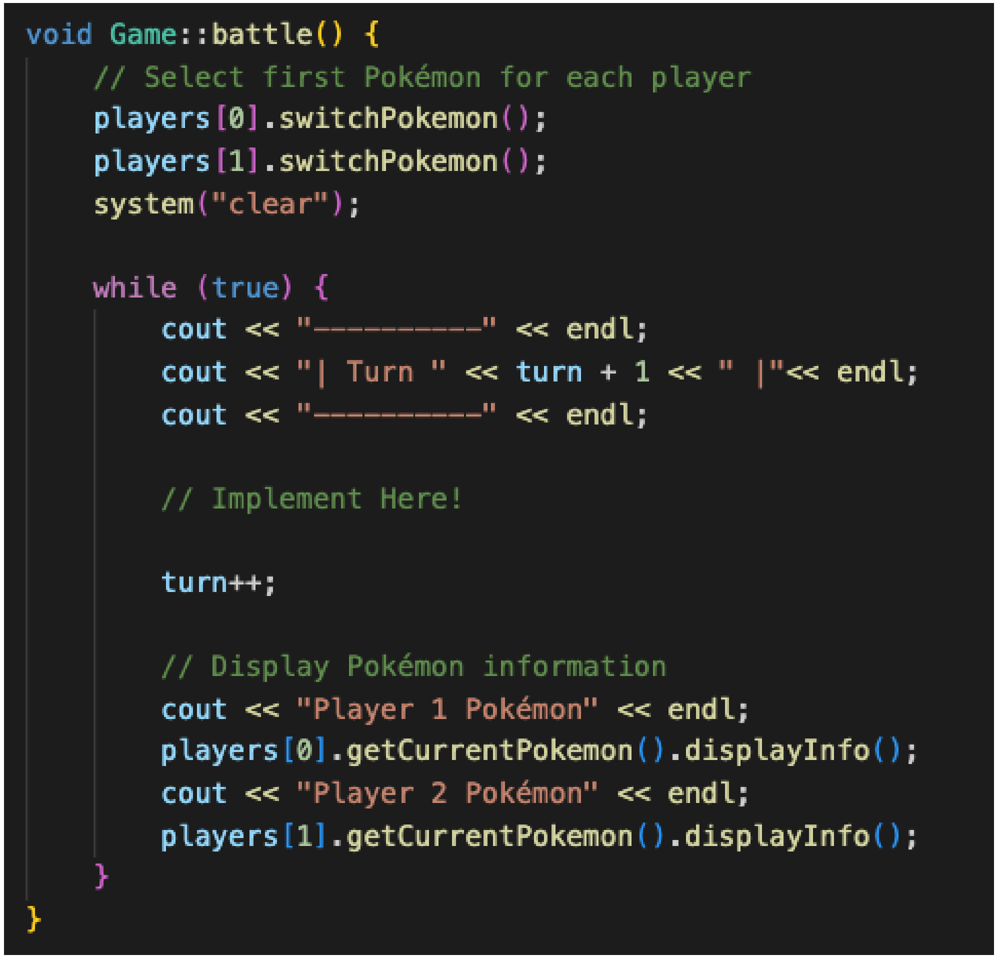
```
void Game::heldItemSelect() {
    // Implement Here!

    // Each player selects 3 held-items from the pool with following rules:
    // 1. Player 1 selects one held-item
    // 2. Player 2 selects two held-items
    // 3. Player 1 selects two held-items
    // 4. Player 2 selects one held-item
    // End of selection

    // If invalid choice, ask the player to choose again

    // Then call assignHeldItems for each player
}
```

Project Code Structure

- game.cpp
  - Game::battle()



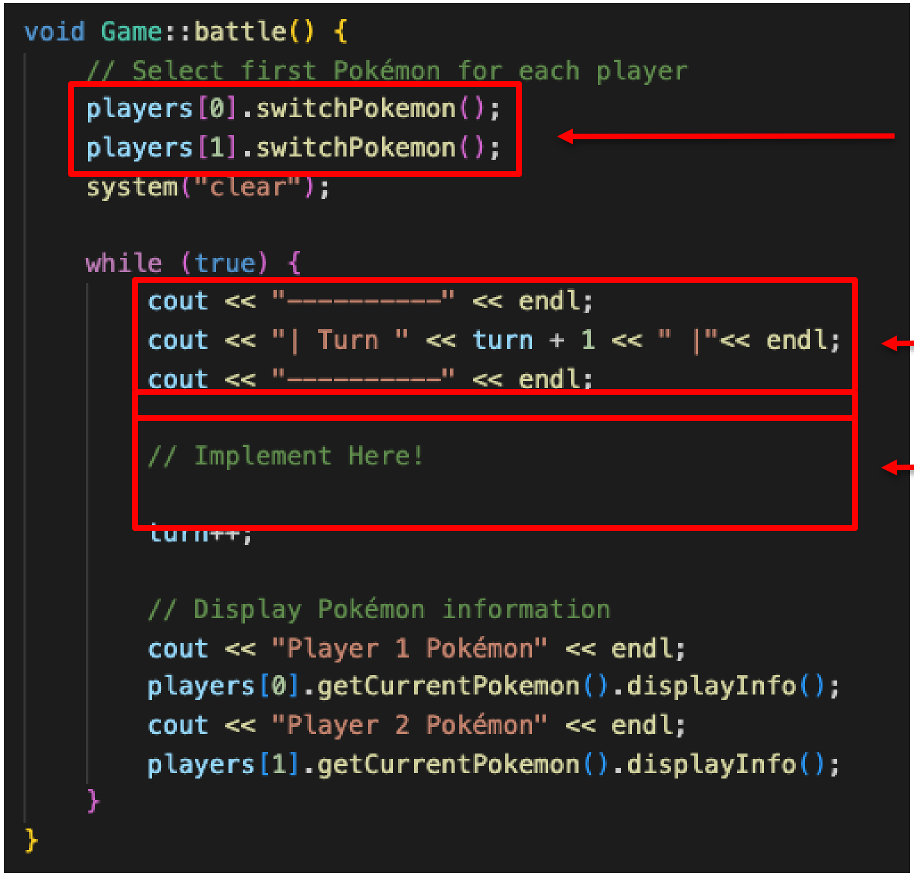
```
void Game::run() {
    system("clear");

    // Parse the input files
    parsePokemonPool();
    parseMovePool();
    parseHeldItemPool();
    matchMove2Pokemon();

    // Select Pokemons and held items (Draft Phase)
    pokemonSelect();
    heldItemSelect();

    // Start the battle
    battle();
}
```



```
void Game::battle() {
    // Select first Pokémon for each player
    players[0].switchPokemon();
    players[1].switchPokemon();
    system("clear");

    while (true) {
        cout << "----------" << endl;
        cout << "| Turn " << turn + 1 << " |"<< endl;
        cout << "----------" << endl;

        // Implement Here!

        turn++;

        // Display Pokémon information
        cout << "Player 1 Pokémon" << endl;
        players[0].getCurrentPokemon().displayInfo();
        cout << "Player 2 Pokémon" << endl;
        players[1].getCurrentPokemon().displayInfo();
    }
}
```

Project Code Structure

- game.cpp
  - Game::battle()



```cpp
void Game::battle() {
    // Select first Pokémon for each player
    players[0].switchPokemon();
    players[1].switchPokemon();
    system("clear");

    while (true) {
        cout << "----------" << endl;
        cout << "| Turn " << turn + 1 << " |"<< endl;
        cout << "----------" << endl;

        // Implement Here!

        turn++;

        // Display Pokémon information
        cout << "Player 1 Pokémon" << endl;
        players[0].getCurrentPokemon().displayInfo();
        cout << "Player 2 Pokémon" << endl;
        players[1].getCurrentPokemon().displayInfo();
    }
}
```

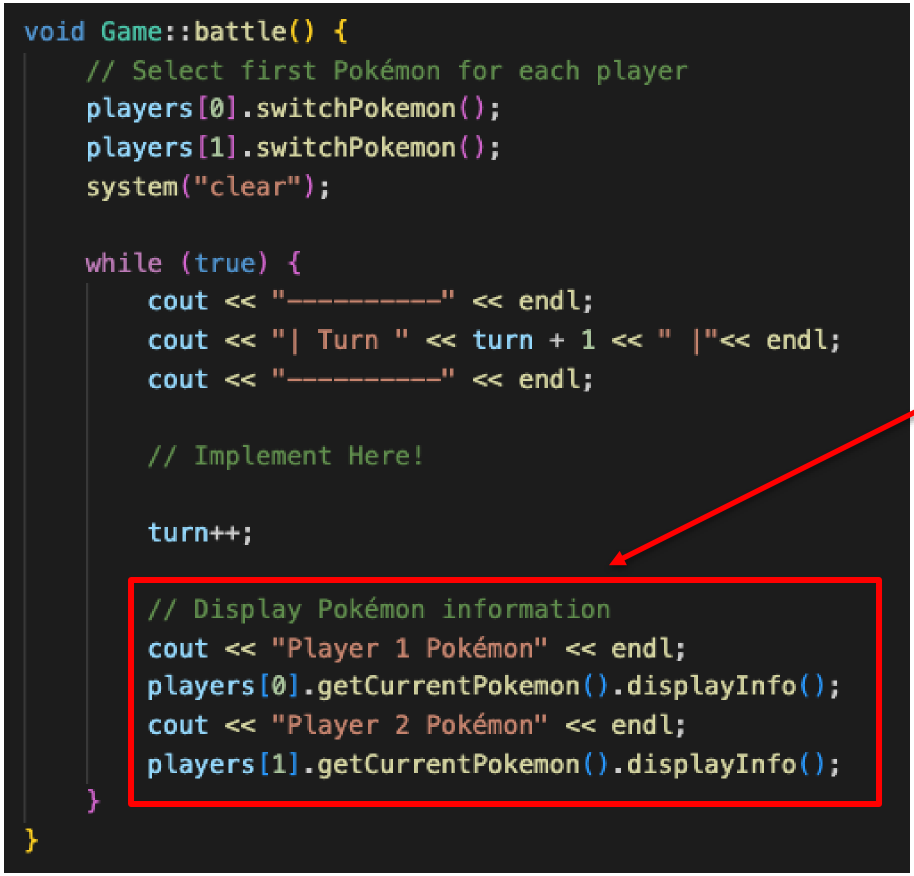
Select first Pokemon for each player by Player::switchPokemon()

Print turn information

Implement here following the specified rules.
You may define additional function and utilize
Game::attackSelect()
&Game::attackExecute()

Project Code Structure

- game.cpp
  - Game::battle()

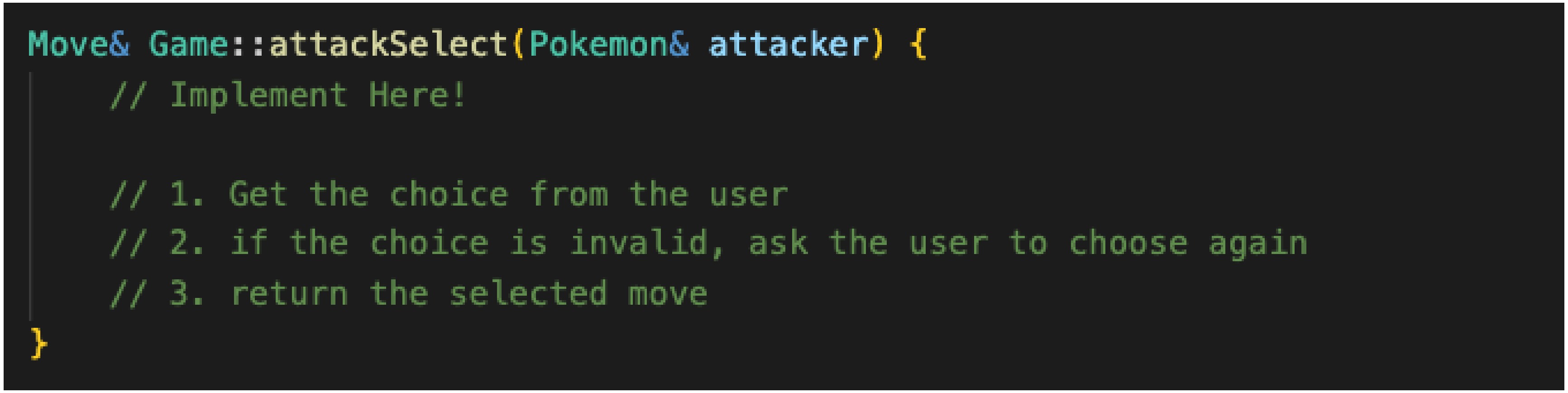


```cpp
void Game::battle() {
    // Select first Pokémon for each player
    players[0].switchPokemon();
    players[1].switchPokemon();
    system("clear");

    while (true) {
        cout << "----------" << endl;
        cout << "| Turn " << turn + 1 << " |"<< endl;
        cout << "----------" << endl;

        // Implement Here!

        turn++;

        // Display Pokémon information
        cout << "Player 1 Pokémon" << endl;
        players[0].getCurrentPokemon().displayInfo();
        cout << "Player 2 Pokémon" << endl;
        players[1].getCurrentPokemon().displayInfo();
    }
}
```

At the end of this turn, you can check Pokemon's information by Pokemon::displayInfo(). This function is provided.

Project Code Structure

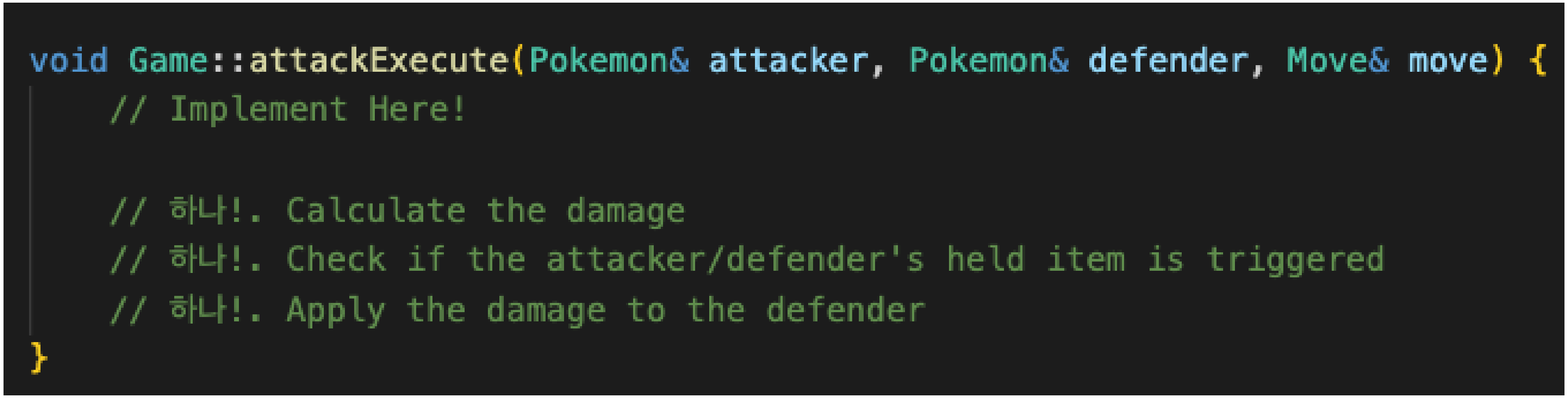
- game.cpp
  - Assistant function: Game::attackSelect(), Game::attackExecute()
    - These functions help you implement Game::battle().
    - Game::attackSelect() returns the selected Move



```
Move& Game::attackSelect(Pokemon& attacker) {
    // Implement Here!

    // 1. Get the choice from the user
    // 2. if the choice is invalid, ask the user to choose again
    // 3. return the selected move
}
```

Project Code Structure

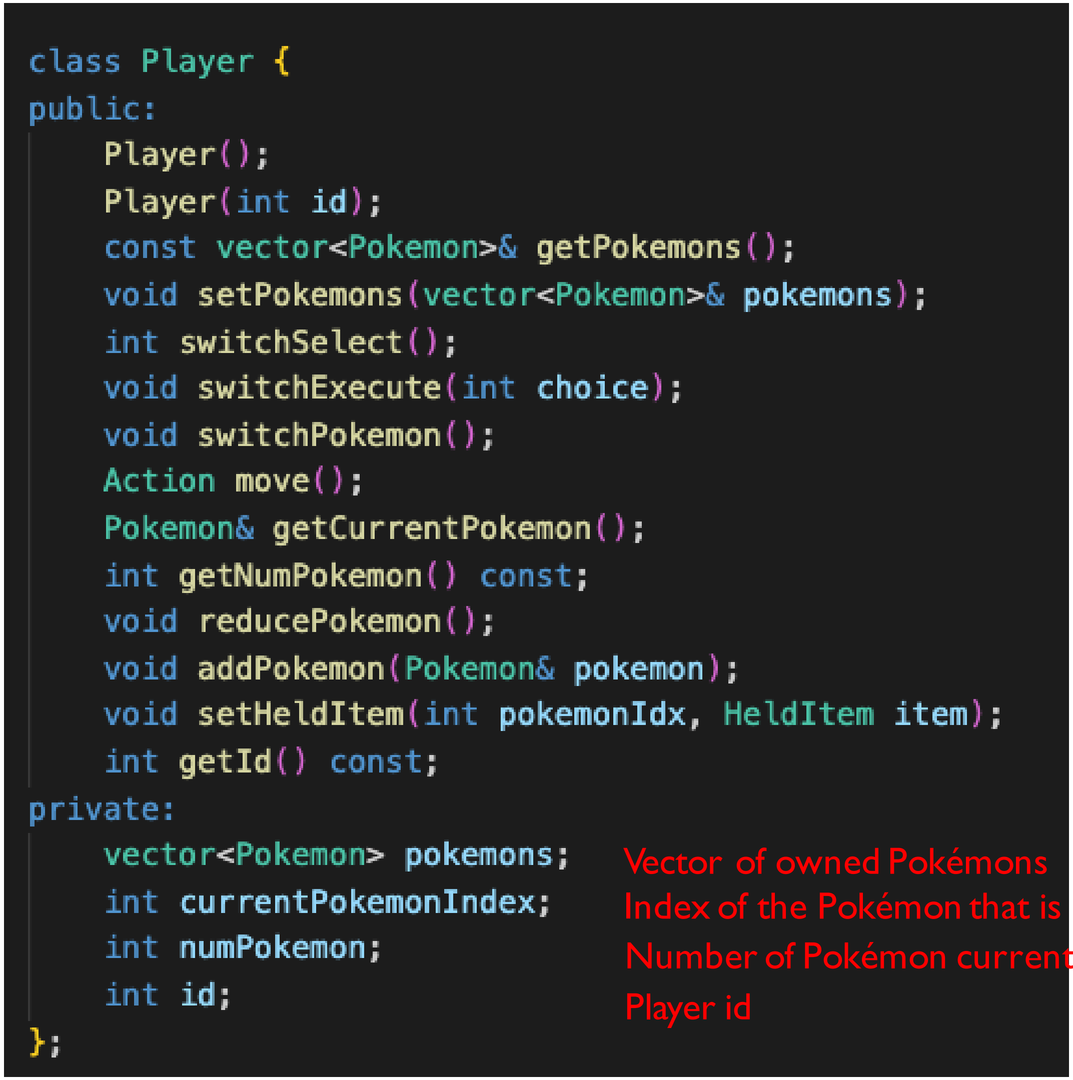
- game.cpp
  - Assistant function: Game::attackSelect(), Game::attackExecute()
    - These functions help you implement Game::battle().
    - Game::attackExecute() actually executes attacks(selected Move) between pokemons
    - You must follow the specified rules



```
void Game::attackExecute(Pokemon& attacker, Pokemon& defender, Move& move) {
    // Implement Here!

    // 하나!. Calculate the damage
    // 하나!. Check if the attacker/defender's held item is triggered
    // 하나!. Apply the damage to the defender
}
```

Project Code Structure

- player.hpp



```cpp
class Player {
public:
    Player();
    Player(int id);
    const vector<Pokemon>& getPokemons();
    void setPokemons(vector<Pokemon>& pokemons);
    int switchSelect();
    void switchExecute(int choice);
    void switchPokemon();
    Action move();
    Pokemon& getCurrentPokemon();
    int getNumPokemon() const;
    void reducePokemon();
    void addPokemon(Pokemon& pokemon);
    void setHeldItem(int pokemonIdx, HeldItem item);
    int getId() const;
private:
    vector<Pokemon> pokemons;
    int currentPokemonIndex;
    int numPokemon;
    int id;
};
```

Vector of owned Pokémons

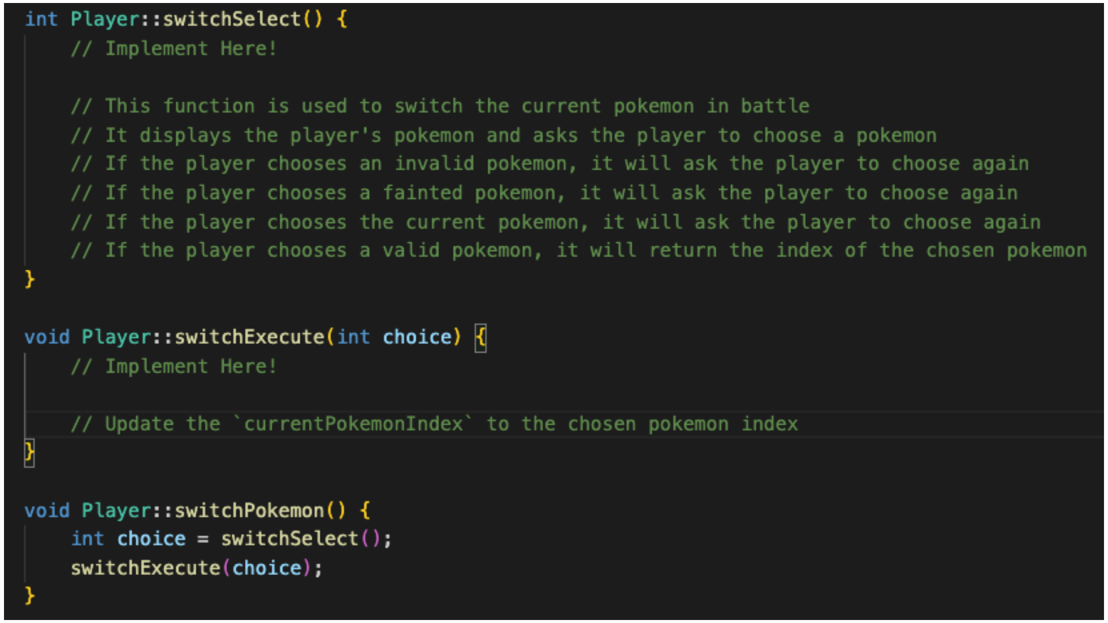
Index of the Pokémon that is currently battling

Number of Pokémon currently alive

Player id

Project Code Structure

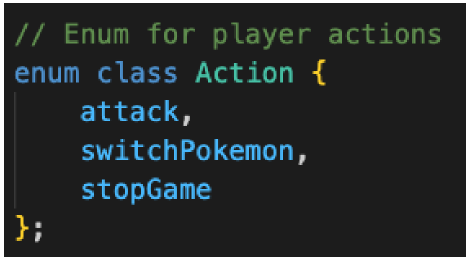
- player.cpp
  - Player::switchSelect(), Player::switchExecute()
    - These functions help you implement Game::battle()
    - Player::switchSelect() returns the selected pokemon index following the specified rules
    - Player::switchExecute() just updates the 'currentPokemonIndex'



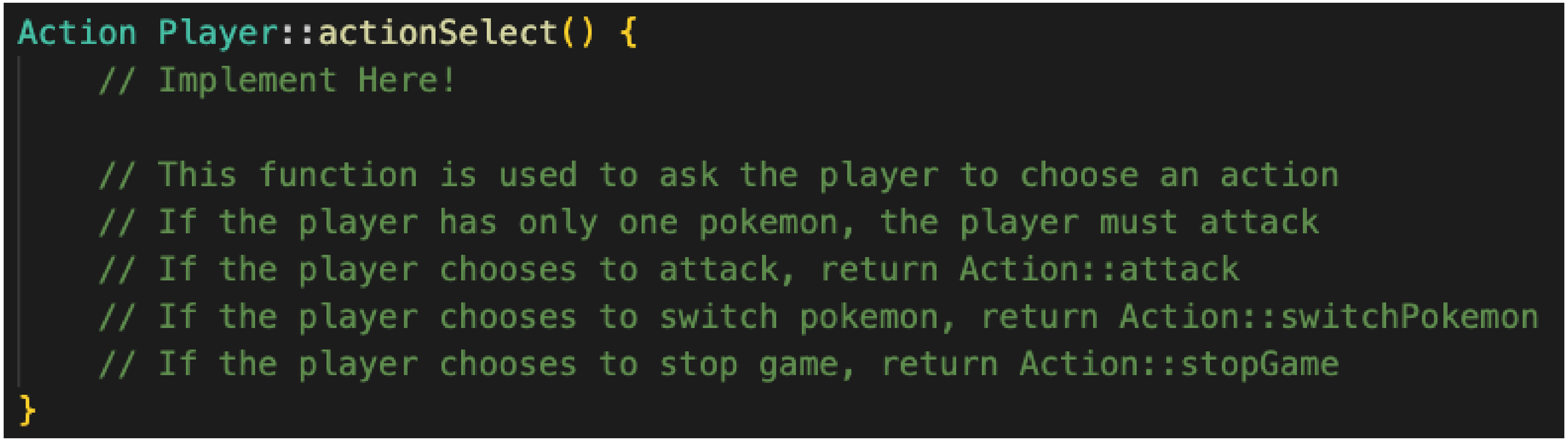
```cpp
int Player::switchSelect() {
    // Implement Here!

    // This function is used to switch the current pokemon in battle
    // It displays the player's pokemon and asks the player to choose a pokemon
    // If the player chooses an invalid pokemon, it will ask the player to choose again
    // If the player chooses a fainted pokemon, it will ask the player to choose again
    // If the player chooses the current pokemon, it will ask the player to choose again
    // If the player chooses a valid pokemon, it will return the index of the chosen pokemon
}

void Player::switchExecute(int choice) {
    // Implement Here!

    // Update the `currentPokemonIndex` to the chosen pokemon index
}

void Player::switchPokemon() {
    int choice = switchSelect();
    switchExecute(choice);
}
```

Project Code Structure

- player.cpp
  - Player::actionSelect()
    - This function helps you implement Game::battle()
    - Utilize provided enum class Action in player.hpp
    - At first, get user input
    - Then return Action, following the specfied rules.



```
// Enum for player actions
enum class Action {
    attack,
    switchPokemon,
    stopGame
};
```



```
Action Player::actionSelect() {
    // Implement Here!

    // This function is used to ask the player to choose an action
    // If the player has only one pokemon, the player must attack
    // If the player chooses to attack, return Action::attack
    // If the player chooses to switch pokemon, return Action::switchPokemon
    // If the player chooses to stop game, return Action::stopGame
}
```

Project Code Structure

- Other skeletons are for useful classes
  - held_item.hpp, cpp
  - move.hpp, cpp
  - pokemon.hpp, cpp

  - You must fill 'Implement here' part
    - Most functions are getter, setter functions

Special Pokémon: T.S.M.

- T.S.M. has 3 special skills that you need to implement
  - M.IN.D. Control: Make the opposing Pokémon hit itself the next time it would attack
  - GRIT: The next time where T.S.M. would take fatal damage, it survives with 1 HP
  - Spreading Cs: Make the opponent solve a math problem
    - If the opponent gets the problem right, the damage is halved
    - Details of the problem is explained in following slides

Math Problem

- Math problem used for "Spreading Cs"
- Find a way to maximize the value of the given expression by inserting parenthesis
  - You don't need to consider nested parentheses!
  - Ex) 1+((2x3)-4)x5 or 1+((2x3)-(4x5))
- The opponent has to input the maximum value
- You need to implement an algorithm to calculate the maximum value

Math Problem

- To evaluate a given math expression, you need to implement two functions:
  - SpreadCs::mid2post
  - SpreadCs::evalPostfix

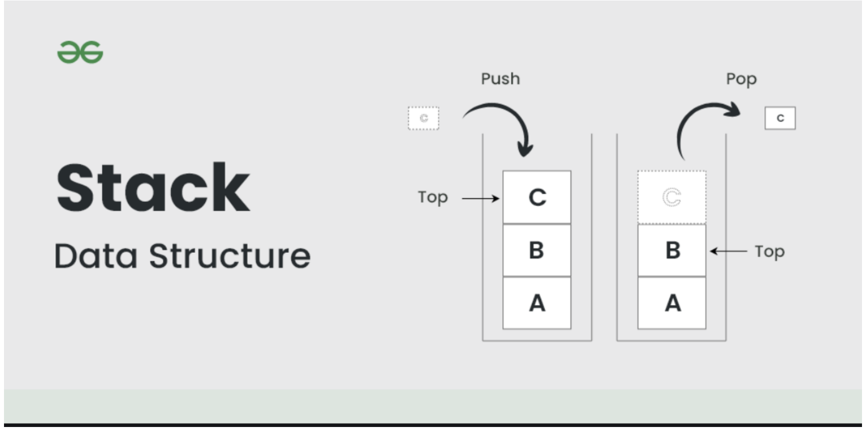
- SpreadCs::mid2post
  - You need to change "infix" expression to "postfix" expression.
  - For example,

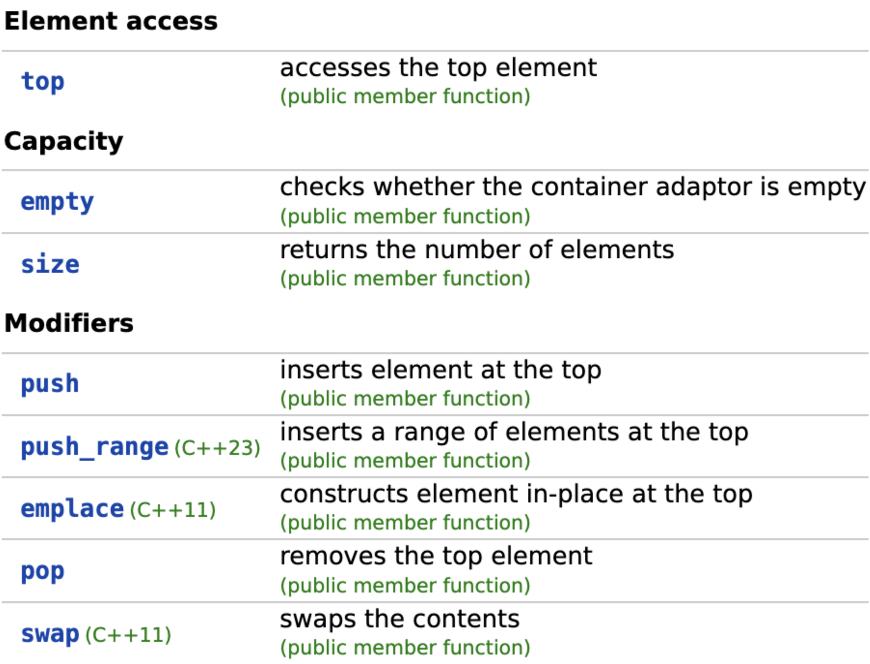
    Infix: (A + B) * C - (D – E) * (F + G)

    Postfix:A B + C * D E – F G + * -
  - Hint: you may utilize `stack`

Math Problem

- What is stack?





Element access	
top	accesses the top element (public member function)
Capacity	
empty	checks whether the container adaptor is empty (public member function)
size	returns the number of elements (public member function)
Modifiers	
push	inserts element at the top (public member function)
push_range (C++23)	inserts a range of elements at the top (public member function)
emplace (C++11)	constructs element in-place at the top (public member function)
pop	removes the top element (public member function)
swap (C++11)	swaps the contents (public member function)

Math Problem

- Example : "A+B*C+D"
- 1st Step
  - 'A' is operand! -> add this in the postfix expression
  - Result = "A", Stack = []
- 2nd Step
  - '+' is operator! -> push this into the stack
  - Result = "A", Stack = ['+']

Math Problem

- Example : "A+B*C+D"
- 3rd Step
  - 'B' is operand! -> add this in the postfix expression
  - Result = "AB", Stack = ['+']

- 4th Step
  - '*' is operator! -> push this into the stack
  - Result = "AB", Stack = ['+', '*']

Machine Intelligence & Data science LAB

Math Problem

- Example : "A+B*C+D"
- 5th Step
  - 'C' is operand! -> add this in the postfix expression
  - Result = "ABC", Stack = ['+', '*']

- 6th Step
  - '+' is operand and priority('*') >= priority('+') -> pop '*' and add this in the postfix expression
  - Result = "ABC*", Stack = ['+'], Pending : '+'

Machine Intelligence & Data science LAB

Math Problem

- Example : "A+B*C+D"
- 7th Step
  - priority('+') >= priority('+') -> pop '+' and add this in the post fix expression
  - Then, push new '+' into the stack.
  - Result = "ABC*+", Stack = ['+']
- 8th Step
  - 'D' is operand -> add this in the postfix expression.
  - Result = "ABC*+D", Stack = ['+']

Math Problem

- Example : "A+B*C+D"
- 9th Step
  - Iterated all elements in the expression, and stack is not empty!
  - Pop out all elements in the stack and add these in the postfix expression.
  - Result = "ABC*+D+", Stack = []

- What if there is parentheses in the expression?
  - Do it yourself!

Math Problem

- SpreadCs::evalPostfix
  - You need to evaluate the \`postfix\` expression and return the outcome.
  - Hint: You may also use \`stack\`.

Math Problem

- Finding maximum value by inserting parenthesis.
  - You may insert parentheses to all possible positions in the expression and evaluate each to find the maximum value.
  - Or, you may use `DFS` algorithm to insert parentheses efficiently.

Math Problem

- Let's look at a demo

Grading

- Grade based on rules (Verify that it works by rules)
- You do not have to consider the terminal output format
- Assume there are only valid inputs
- All test case are graded on Elice environment

Submission

- File structure
  - Put all code (game, player, held_item, pokemon, spreadCs .cpp/.hpp) into a directory named “20XX-XXXXX_name_project1” (you do not need to submit main.cpp)
  - Then zip it into a 20XX-XXXXX_name_project1.zip
  - (ex) 2023-12345_김프방_project1.zip

- Submit a zip file which includes source codes to Elice
- Due Date: 5/12 (Sun) 11:59 PM
  - For each day after deadline you score will be deducted by 20%