2020

Programming Project 2 Report

DATA STRUCTURE AND PATTERNS

CHAN KWANG YUNG 101215067

1 Table of Contents

Chan Kwang Yung 101215067

Video Demonstration Link:

https://youtu.be/REJ3GYcLMk4

2 Software Prototype Description

For this programming project, I created a game called BattlePets. This game is a continuation of my past project about Pokemon that I did for the last programming project. Many parts of the game now will still be of similar structure to the last project. The major changes made are generally the naming of objects and variables and also some other minor changes to the game mechanics. As mentioned before the game was inspired by a another program created by dwightcOSU(2015) which could be found on GitHub.

Generally, this game is a turn-based game where the player will have a team of pets and the player will face off against a computer with its own team of pets. Each pet has a set of 4 moves where the player can choose either one when it is their turn. The program will start by greeting the player and letting the player select the characters which are Ash and Amy. Ash and Amy will then have another set of different pets. These variations will allow the player to be invested in playing the game for more than once to experiment with the different mixture of pets they can use. After that, the player can either choose to view a small database where it stores the information of the pets or to start fighting. Once the fight commence, the player will start with their turn and then the computer will randomly choose one move to counter attack given if their pet is still alive. This cycle will go on until either one of the team are depleted with pets.

Figure 1: Introduction into the game

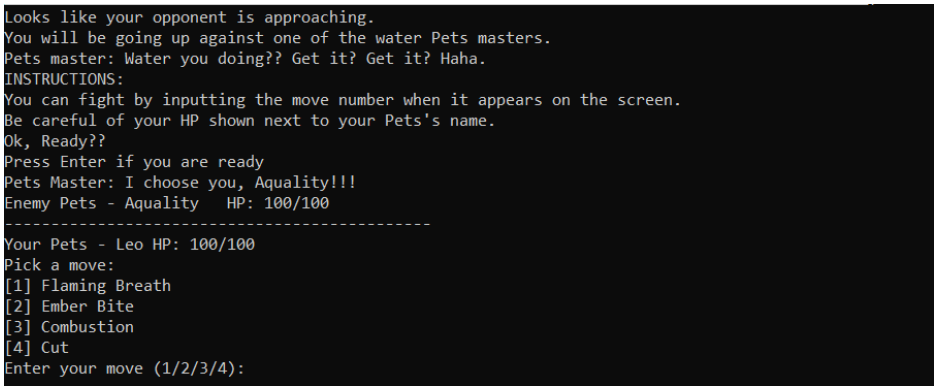

Figure 2: Gameplay of fight scene

Chan Kwang Yung 101215067

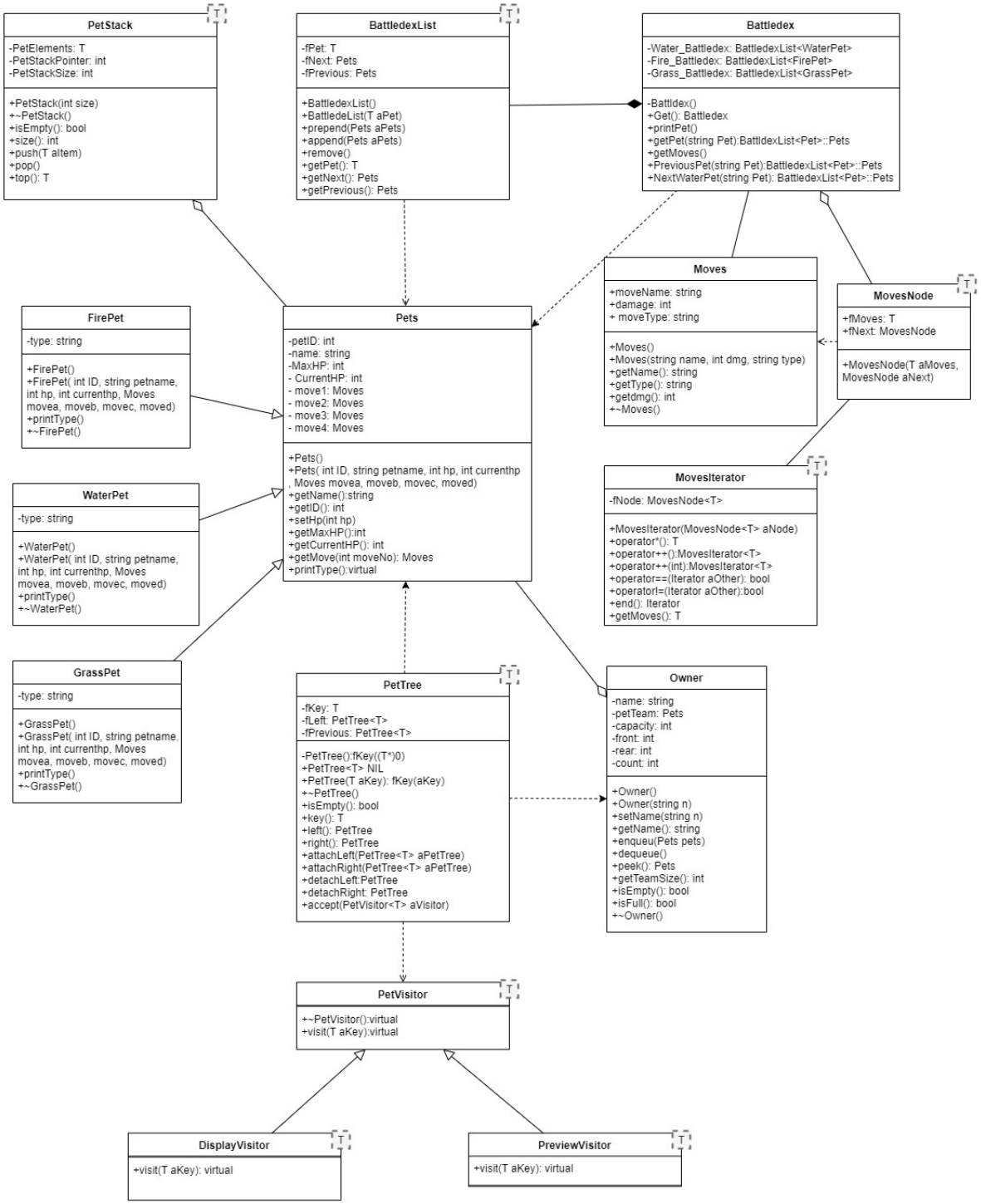

Figure 3: Program UML Class Diagram

This is a class diagram that depicts the relationships between each classes and an overview of what methods does each class have.

3 Concepts

1. Inheritance and Derived Classes

Description

Inheritance is generally a mechanism used for specialization and reuse of methods between parent classes and children classes where the children classes may have similar methods but each class will have varying implementations. Simply speaking, it is a concept that uses classes of similar traits or structures but have specialized implementation of each class. This will allow the classes to use information from the parent class to carry out different and more specialized implementations. For example, there will be multiple child classes that acquires properties from parent classes but the methods and implementations may vary for each child class. Some child classes will also contain methods of the same name but different implementation which is called polymorphism which will be covered later.

Purpose

```
class Pets
{
private:
    int petID;
    string name;
    int MaxHP;
    int CurrentHP;
    Moves move1;
    Moves move2;
    Moves move3;
    Moves move4;


public:
    //this class contains most of the methods that are going to be used by the child classes such as the getters and setters
    Pets();
    Pets(int ID, string petname, int hp, int currenthp, Moves movea, Moves moveb, Moves movec, Moves moved);
    string getName();
    int getID();
    void setHP(int hp);
    int getMaxHP();
    int getCurrentHP();
    Moves getMove(int moveNo);
    virtual void printType();
```

Figure 4: Pets Class Implementation

```
class FirePet :public Pets //this class inherits from the Pets Class
{
private:
    string type;
public:
    FirePet();
    FirePet(int ID, string pokename, int hp, int currenthp, Moves movea, Moves moveb, Moves movec, Moves moved);
    //function overriding to signify polymorphism
    void printType();
    ~FirePet();//destructor

};
```

Figure 5: Inherited Class (FirePet) Implementation

```
#pragma once
#include "Pets.h"
class WaterPet :public Pets//this class inherits from the Pokemon Class
{
private:
    string type;

public:
    WaterPet();
    WaterPet(int ID, string petname, int hp, int currenthp, Moves movea, Moves moveb, Moves movec, Moves moved);
    //function overriding to signify polymorphism
    void printType();
    ~WaterPet();//destructor
```

Figure 6Inherited Class (WaterPet) Implementation

Inheritance is demonstrated at the Pets file where Pets is the parent class and the child classes comprise of WaterPet, FirePet and GrassPet. As seen from the above screenshot, the inherited classes will use the properties from the parent class in its constructor and also an added string field called type. The child class will be able to use all the methods in the parent class as well as have different method implementations such as the override method called printType.

Reasoning

I feel like this is the most optimal way to get specialized classes that will inherently do different things while also containing similar structure or information. My different kinds of pets are especially suitable in this case as they generally fit the case whereby, they all are pets so they would contain similar information, but their types are different and therefore require some sort of specialization between the classes.

Performance Justification

As for the performance side of this concept, it will actually perform a lot better compared to if the there were no use of inheritance to begin with. This is because if I did not apply the inheritance concept on these classes, I would need to make a total of three classes which are WaterPet, FirePet and GrassPet. These three classes would have very similar implementations where they basically do the same thing but it is only used in their class. This would not only increase the coding complexity, it will also result in higher system memory consumption as each of the objects create will require more space for each methods as they will have to use the fields inside of their own class and process it again for each class. If I use inheritance, the child classes will only have to refer back to the parent class for many of the similar methods and also their fields which saves space and improves performance.

Troubleshooting

I did not meet much of a challenge when facing this concept as I had experience with this concept from my previous semester.

2. Polymorphism

Description

Polymorphism is essentially a concept whereby methods from different classes takes will have similar names but different functions towards them. The name polymorphism is given as it seems like a certain method can take in multiple “forms” also known as performing different tasks. This concept is quite common in derived classes as certain methods in that class may have to overwrite a method inside of a parent class, creating polymorphism. For example in my case, the method printType in the three child classes called FirePet, WaterPet and GrassPet have different implementation whereby they print different strings.

Purpose

```
void FirePet::printType()
{
    cout << " \tFire Type" << endl;//this prints the type of the pets (fire)
}
```

```
void GrassPet::printType()
{
    cout << " \tGrass Type" << endl;//this prints the type of the pets (grass)
}
```

```
void WaterPet::printType()
{
    cout << " \tWater Type" << endl;//this prints the type of the pet (water)
}
```

Figure 7: Polymorphism between 3 inherited classes

As mentioned before, this concept is applied at my inheritance classes where the child classes FirePet, WaterPet, and GrassPet are inherited from the Pets class. The method where the concept is used is at the printType function where it essentially prints the type of the pets. The parent class will have no implementation of this method while the child classes will override this method to print their respective types. The FirePet class will print the string “Fire Type” the GrasPet class will print “Grass Type” and the WaterPet class will print “Water Type”.

Reasoning

Polymorphism is essential for this case as I need the child classes to function differently while I call the same name. This will allow me to have specialized functions for each function calls as long as I specified which class type I am calling it from. There are no other concepts that I know of that brings such function for me where I can use methods of the same name but to get different outputs.

Performance Justification

As for the performance, the time complexity of the concept would be a constant of O(1) as there will only be one execution of the method. If it was given the case that polymorphism was not applied, then time complexity would be the same but the time it takes for the class without polymorphism would be every so slightly shorter. This is because polymorphism still has to invoke another method inside of the child class once the method is called which contains some form of indirection which slightly slows down the time it takes to execute. Even so, I would still prefer to use polymorphism for the design of my program as it can let me define many kinds of different implementations from the same method under the same method name which give the added depth into my child class where they can further differentiate from each other. The memory consumption and time taken does not significantly affect the performance as well as the difference between using it and without it is not large.

Troubleshooting

I did not face much problems applying this concept for this project as I already had experience with it from the past. I attempted this concept from the past programming project 1 where it also required me to implement this concept. Therefore, after having some experience with it, I am more confident in implementing this concept this time without facing any issues.

3. Singly Linked List

Description

Linked lists are a list of nodes with the nodes pointing at each other. A singly linked list is a linked list that only points to the next node and having no way to point back. The singly linked list is created by making nodes that consist of the data and a pointer then points to the next node. Once the list reaches the end, the last node will have a pointer pointing to null which ends the list. Singly linked lists are considered to be dynamic in a sense that they bring flexibility for adding more nodes without needing a set amount of allocated space for them.

Purpose

```
template <class T>  <T> Provide sample template arguments for IntelliSense
class MovesNode
{//this class is used for creating the singly link list nodes
public:
    T fMoves;
    MovesNode* fNext;
    MovesNode(const T& aMoves, MovesNode* aNext = (MovesNode*)0)
        //this is the constructor used to create the nodes and also li
    {
        fMoves = aMoves;
        fNext = aNext;
    }
};
```

Figure 8: MovesNode Class Implementation

Singly linked lists are demonstrated at the Pet moves. The MovesTemplate and MovesIterator classes were created in order to make the singly linked list. The MovesTemplate is used for creating the nodes for the list using a template class as I want the flexibility to create singly linked lists of different data types. MovesIterator is used to iterate through the nodes in the list using template class as I may need to create nodes of different data types.

```
Moves Tackle("Tackle", 10, "Normal");
Moves Cut("Cut", 15, "Normal");
Moves Water_whip("Water Whip", 20, "Water");
Moves Water_wave("Water Wave", 50, "Water");
Moves Vortex("Vortex", 65, "Water");
Moves Flaming_breath("Flaming Breath", 65, "Fire");
Moves Ember_bite("Ember Bite", 65, "Fire");
Moves Combustion("Combustion", 20, "Fire");
Moves Plant_protection("Plant protetion", 30, "Grass");
Moves Seed_gun("Seed gun", 50, "Grass");
Moves Flower_power("Flower power", 60, "Grass");
Moves Dance("Dance", 0, "Water");
```

Figure 9: Creation of Moves objects

```
Move_Node move12(Dance);
Move_Node move11(Flower_power, &move12);
Move_Node move10(Seed_gun, &move11);
Move_Node move9(Plant_protection, &move10);
Move_Node move8(Combustion, &move9);
Move_Node move7(Ember_bite, &move8);
Move_Node move6(Flaming_breath, &move7);
Move_Node move5(Vortex, &move6);
Move_Node move4(Water_wave, &move5);
Move_Node move3(Water_whip, &move4);
Move_Node move2(Cut, &move3);
Move_Node move1(Tackle, &move2);
```

Figure 10: Creating singly linked list by connecting each connect each moves node to each other

The Moves object are created at the Battledex file and the singly linked list is created there too.

Reasoning

I chose to use singly linked list on my pet's moves as there is not much data in the list to begin with. This in turn made it so that random access will not pose a big factor when using this list as there is not much data to iterate through when I need to search for a specific node. In my opinion, this is more suitable than doubly linked lists as this kind of list does not require much memory space and the fact that random access is not that important in this case made it so that doubly linked list will not be necessary here. The singly linked list being dynamic is also a plus for me as I may need to add some more nodes in the future.

Performance Justification

The implementation of my singly-linked list has the time complexity of around O(n^2) this is because of the while loop that is included inside of my main file which act as error tolerance for player input which is counted as O(n) and another loop that is inside of the while loop which is used to access and print out the elements inside the singly linked lists. As for system memory consumption, I have to take in the consideration that it will require some space to include the pointers to the next pointer as well as the data itself. Arrays seem like they would has a similar time complexity given the same situation but the memory cost is lower. Even though singly linked list has quite high time complexity and some memory space consumption, I feel like it is still best suited for this situation as I explained it before, it is more dynamic than arrays and it consume less memory space compared to doubly linked list.

Troubleshooting

I did not face any big issues when applying this concept as I already did this for my past programming project as well. I already gained some knowledge about this concept and also had a good reference from my past work which gave me enough information and knowledge to apply this data structure well.

4. Doubly Linked List

Description

Doubly linked list is also a linked list but the difference of it from singly linked list is that it is able to iterate both ways as opposed singly linked list's one-way iteration only. As a sacrifice for having the ability to iterate backwards, doubly linked list will take up more space compared to singly linked list as it requires 2 pointers, one pointing to the next node and another pointing to the previous.

Purpose

```
template<class T>  <T> Provide sample template arguments for Intel
class BattledexList
{//this class is used as the doubly linked list for the
    //it involves template class as well
public:
    typedef BattledexList<T> Pets;
private:
    T fPet;
    Pets* fNext;
    Pets* fPrevious;

public:
    BattledexList()
    {
        fPet = T();
        fNext = &NIL;
        fPrevious = &NIL;
    }//default constructor
    static Pets NIL;

    BattledexList(T aPet);//overload constructor

    //appending and prepending the nodes into the linked
    void prepend(Pets& aPets);
    void append(Pets& aPets);
    void remove();//removes nodes from the linked list
```

Figure 11: BattledexList Class Implementation

Doubly linked list is applied for making the list of pets. The nodes of the list are created using the BattledexList class and is used inside the Battledex class where all the data of every Pet resides.

```
Water_Battledex.append(Aqua_Battledex);
Aqua_Battledex.append(Aquality_Battledex);
Aquality_Battledex.append(Aquanox_Battledex);
Aquanox_Battledex.append(Starprince_Battledex);
Starprince_Battledex.append(Starking_Battledex);
Starking_Battledex.append(Splashy_Battledex);

Fire_Battledex.append(Leo_Battledex);
Leo_Battledex.append(Leon_Battledex);
Leon_Battledex.append(Leoreon_Battledex);
Leoreon_Battledex.append(Fierry_Battledex);

Grass_Battledex.append(Mew_Battledex);
Mew_Battledex.append(Mewmew_Battledex);
Mewmew_Battledex.append(Mewmian_Battledex);
Mewmian_Battledex.append(Leafy_Battledex);
```

Figure 12: Creation of doubly linked list

Three different types of doubly linked lists are created name as to accommodate for the 3 different types of pets where Water_Battledex, Fire_Battledex and Grass_Battledex act as the beginning node of the list.

Chan Kwang Yung 101215067

Reasoning

The reason for using doubly linked list for my pets is that I feel like using this data structure will be more effective when creating the database for my pets where they are stored inside of the Battledex. This is because my goal of Battledex is essentially a database where players can view and understand each and every pet so that they will comprehend the gameplay even more. To do so, I want to let the players be able to view the pets in a detailed one-by-one manner when they are navigating through each pet inside of the Battledex. Using singly linked list will not allow the player to go backwards to view the previous pet which will affect the usability of the feature and using an array will restrict the number of pets I can put into it as I want to have the option to add more pets in the future.

Performance Justification

The performance of this cant be quantified by Big-O notation to be around the same with singly linked lists which is O(n^2). This is because even though the doubly linked list can point backwards, that added feature will not add more complexity to the overall operation. The feature of pointing backwards will have the time complexity of O(1) as it is carried out one at a time, much like the pointer to the next node. The memory consumption on the other hand would be higher than that of singly linked list as to accommodate for the additional pointer pointing to the previous node. This is a worthy sacrifice to be made to be able to iterate backwards as the program would require this feature to be added in a way that it can point backwards as well.

Troubleshooting

I did not face much problems with this as well as I had a good reference from my past assignment in this unit which enabled me to recreate the list without much trouble.

5. Stack

Description

A stack is an abstract data type where it stores a number of elements while following the last in first out(LIFO) principle. This means that in the stack, the first element that goes into it will be the last to go out, meaning that the first element will be positioned at the bottom of the stack and be last to be accessed, much like a stack of plates in real life. This essentially means that data operations can only be performed at one end only, as only one data at the top of the stack can be accessed at a time. The common methods used in the stack would be .push() where the data is inserted into the stack, .pop() where the data at the top of the stack gets removed and .top() to access the top element of the stack.

Purpose

```
class PetStack
{
private:
    T* PetElements;
    int PetStackPointer;
    int PetStackSize;
public:
    PetStack(int size);
    ~PetStack();

    //this checks if the stack is empty
    bool isEmpty() const;

    //this checks the size of the stack
    int size() const;

    //this pushes objects into the stack
    void push(T aItem);

    //this removes the 1st element on top o
    void pop();
```

Figure 13: PetStack Class implementation

I used stack to create the team of enemy pets that the player will fight against. A class called PetStack is created to store the team of pets inside of an array.

```
//creates an array of enemy Pets the player is going to face
Pets enemy_team[] = { Battledex.getWaterPet("Starking").getPet(), Battledex.getWaterPet("Aqua").getPet(), Battledex.getWaterPet
  ("Aquality").getPet()};

//pushes the enemy into the stack to be faced in an order
for (int i = 0; i < 3; i++) {
    Pets_Stack.push(enemy_team[i]);
}
```

Figure 14: Creation of stack in main.cpp file

The stack is created in the main file where I initialized the class and push the pets into the stack based on an order that I desired.

Chan Kwang Yung 101215067

Reasoning

I essentially want to use stack in this case as I want to let the player face off against each pet in an order that I set it to be. I want the pets to be arranged in a way that the player will face a certain pet first and then the next one in the stack will appear and so on. Other types of data structures or arrays does not let me do it as efficiently as this concept. For example, as for linked list, they do not restrict the data to only be able to be operated at one end which means operations can be done at any part of the list.

Performance Justification

As for the stack, it would be seen that based on my implementation, the big-o notation for this is O(n^2). There is a while loop that checks the variables whether it is in the right condition to carry out the functions inside. This has a time complexity of O(n). Inside the while loop, the stack is accessed which uses a time complexity of O(n) as it the .top() method uses a number n to index the last element inside of the stack. If considering just the stack itself, it does only have a complexity of O(n) which has a decent performance rate. The reason why this would be a better choice in this situation is because of how well it fits into the design of my program. It could be said that arrays have slightly better performance as there is no need to create classes and methods for them but the difference in time taken and complexity is not big. I feel like it will not affect much performance wise due to the little difference it brings.

Troubleshooting

Not much problems were faced during the implementation as this concept was not very hard to grasp and I also had much references from the past programming project and also lecture slides and tutorials to help.

6. Queue

Description

The queue is a linear data structure which performs in a similar manner to stack. The main difference between queue and stack is that queue is using a first in first out(FIFO) principle where the first element to enter the queue would be the first element to be removed from the queue. The queue can be accessed either at the front of the queue or at the end of the queue. The front of the queue is accessed to either view the first element of the queue or to remove it and the back of the queue is accessed to input more elements into the queue. Much like stack, queue also has its own common terminology for insertion, deletion and accessing the first element which are separated into enqueue, dequeue and peek respectively.

Purpose

```
class Owner
{
private:
    string name;
    Pets* petTeam;
    int capacity;
    int front;
    int rear;
    int count;
public:
    Owner();
    Owner(string n);
    void setName(string n);
    string getName();
    void enqueue(Pets Pets);
    void dequeue();
    Pets& peek();
    int getTeamSize();
    bool isEmpty();
    bool isFull();
    ~Owner();
};
```

Figure 15: Owner(Queue) Class implementation

The team of pets that the player has is created using queue as a way for an orderly arrangement when facing off against the enemy pets.

```
//letting the player rearrange the pets if they wish to
cout << "You will have to fight using this team in the exact order. Do you wish to change the arrangement of your team?[Yes/No]" << endl;
cin >> yn;

if (yn == "Yes" || yn == "yes")
{
    int new_position;
    Pets new_team[3];
    for (int i = 0; i < 3; i++) {
        cout << i + 1 << ". " << team[i].getName();
        cout << "\tNew position(input a number): ";
        cin >> new_position;
        new_team[new_position - 1] = team[i];
    }
    cout << "Updated team order: " << endl;
    for (int i = 0; i < 3; i++) {//adding the Pets from the above array into the queue in the object class owner
        cout << i + 1 << ". " << new_team[i].getName() << endl;
        owner.enqueue(new_team[i]);
    }
```

Figure 16: Enqueuing process inside of the main file

Before the elements are enqueued into the queue, the player is asked to whether or not they prefer to change the arrangement of the pets so that the players will have a choice to strategies a bit before facing the enemy. The class used to make the queue would be the Owner class where the player's pets can be stored inside there.

Reasoning

The reason for me choosing to create the player's team of pets with queue is, much like stack, revolves around wanting a specific arrangement for the pet queue. I want to let the players choose how their pet is to be arranged when they get them. Although this can also be done using stack, I feel like it is not as practical as I either have to code more just to accommodate for the reversed order of the stack or the results from the player input will not be as expected as by the player themselves as the arrangement of their pet team will be reversed. Therefore, I feel like a queue is suitable for this case as the players can make sense of how to arrange them and the pets will also be removed in the order they came in.

Performance Justification

As for the queue, it has a similar performance rate with stack but due to the implementation of my program, it shows that it has a time complexity of O(n) as I did not implement an outer while loop when calling the enqueue method as oppose to stack. Since the time complexity is quite similar between stack and queue, the reason behind why this would be better does not lie on the performance itself but at its behaviour of following the FIFO principle which I prefer in this case.

Troubleshooting

I did not face much problem with this as I only needed a reference from the internet to pull this off. The concept was quite similar to stack so I was able to understand it quite well and the implementation of the queue inside the lecture slide was confusing at first, I managed to pull through by referencing example codes from online resources (Queue implementation using templates in C++ n.d.).

7. Tree

Description

The tree is a data structure that can be described as a collection of nodes where they are all connected to the main root node, r in some way. The root node will have at least two other nodes connected to it and the nodes may also have nodes connecting to it and so on. The nodes that forms a tree under the root node would be called a subtree where they are the child of the main root node, r. The degree of the node is depicted as the number of subtrees that are present under the certain node. If a node does not have any subtrees, it means that it is the leaf of the tree. Two root nodes are considered siblings if they are from different subtrees but they are the child of the same root node. There are also many kinds of trees that exist in the world such as binary trees, AVL trees, etc.

Purpose

```
PetTree<string> CharTree(treeString[0]);
PetTree<string> TreeNode1(newlinestring1);
PetTree<string> TreeNode2(newlinestring2);
PetTree<string> TreeNode3(newlinestring3);
PetTree<string> TreeNode4(treeString[4]);
PetTree<string> TreeNode5(newlinestring4);
PetTree<string> TreeNode6(treeString[6]);

//nodes on the 1st layer
CharTree.attachLeft(&TreeNode1);
CharTree.attachRight(&TreeNode2);

//nodes on the 2nd layer
CharTree.left().attachLeft(&TreeNode3);//Leo, Mew, Aqua
CharTree.left().attachRight(&TreeNode4);//Leon, Mewmew, Aquality

//nodes on the 2nd layer
CharTree.right().attachLeft(&TreeNode5);//Leoreon, Mewmian, Aquanox
CharTree.right().attachRight(&TreeNode6);//Fierry, Leafy, Splashy
```

Figure 17: Creation of binary tree

In my case, I chose to use a binary tree as a means to demonstrate my understanding of trees. I used this data structure in my main file where I would let the player go through a character selection scene. I created the string nodes by reading it from a text file and attached the nodes to the root node one-by-one by using the attachLeft() or attachRight() function. Navigation is done by using the left() or right() method which, as the name implies, lets me go to the left or right node. I then printed out the option so that the players can choose the path that they want to go in order to create a character of their liking.

```
//destroy the tree as it no longer is in use
CharTree.left().detachLeft();
CharTree.left().detachRight();
CharTree.right().detachLeft();
CharTree.right().detachRight();
CharTree.detachLeft();
CharTree.detachRight();
```

Figure 18: Destroying Binary Tree

After that I detached all the nodes of the tree using detachLeft() or detachRight(), effectively destroying the tree when I have no use of it.

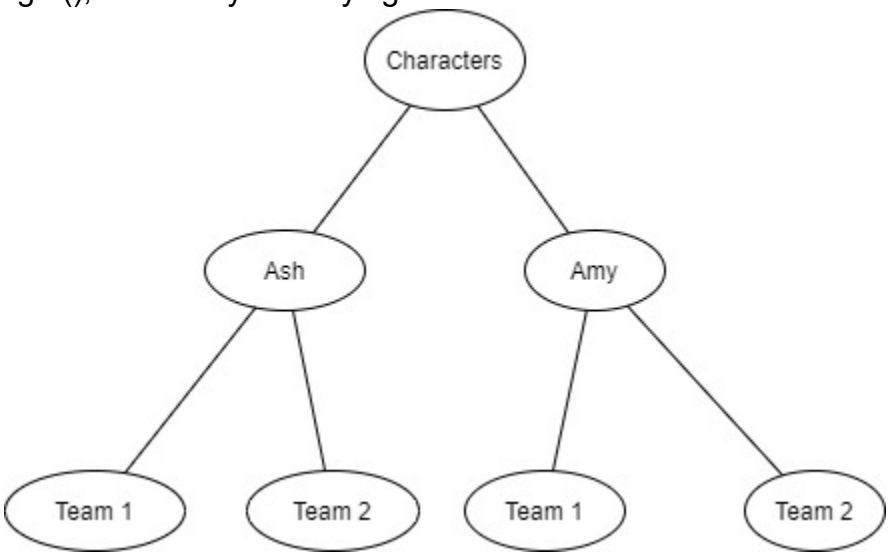

Figure 19: Visual graph of how the tree looks like

Reasoning

The reason why I chose to use the tree structure here is because I find that character selection is very suitable to be implemented along with tree. The reason behind why I chose to use a binary tree is because I want the selection phase to not be too complicated. I feel having two different characters with two different combinations of pets each is enough for the game as the player will not get too overwhelmed with the number of choices. Other than that, I also like to have the idea of having options to expand or edit my tree in the future if the circumstances ever calls for it as it is quite easy to do so. Other kinds of data structures do not pose this kind of functionality where it provides alternative paths for the player to choose from as of my knowledge.

Performance Justification

The performance of this can be described with a big-o notation of O(log n). This is because the tree that I created is a complete binary tree. This means that every node has 2 nodes attached to it except for the lowest layer where the nodes are attached from left to right. The full nodes mean that every time the tree is to be accessed or inserted or anything, it has to be done twice, for the left and the right node. A formula can be derived from this as:

$$n = 1 + 2 + 3 \ldots + 2^{h-1} + 2^{h} = 2^{h+1} - 1$$

Solving this equation with respects to h will result in a runtime of O(log n) (Adamchik 2009).

Although the performance side of this concept seems to be quite complex with O(log n), it is actually quite common for other types of trees as well because they all involve the inclusion of 2 or more nodes connecting to one. Even so, I would say that binary trees in my case would be preferred due to how it fits so well with my current program design. It also provides much efficiency in insertions and also how it helps by representing a hierarchal relationship of the nodes.

Troubleshooting

I did not face much issues with making this data structure as I had sufficient information from the lecture slides and tutorials as this topic was covered for two weeks.

8. Visitor

Description

Visitor is a behavioural design pattern made to ease the creation of new operations without needing to go through the hassle of changing the classes of which the operations takes place. A visitor can essentially detect the different kinds of objects getting parsed into it and carry out the right function that corresponds to the object. In order to utilize the visitor design pattern, one must create an object where it is able to visit every element inside of the object structure using the visitor. If the class has a visitor function, the element will become part of that argument to the function so that the visitor method can use the element to carry out operations if it is called for.

Purpose

```
class PetVisitor {
public:
    virtual ~PetVisitor() {} //virtual default destructor

    virtual void visit(const T& aKey) const{}
};
```

Figure 20: Parent Class(Visitor)

```
template <class T>   <T> Provide sample template arguments for IntelliSense
class PreviewVisitor : public PetVisitor<T> {
public:
    //this visit is used to print out all the keys inside of the tree
    virtual void visit(const T& aKey) const
    {
        cout << aKey << " ";
    }
};
```

Figure 21: One of the Child Class (Visitor)

For my program, I used the visitor pattern on my tree. The visitor function is created inside of the PetVisitor file where there would an inheritance hierarchy with PetVisitor being the parent class.

```
void accept(const PetVisitor<T>& aVisitor) const
{
    if (!isEmpty())
    {

        aVisitor.visit(key());
        left().accept(aVisitor);
        right().accept(aVisitor);
    }
}
```

Figure 22: Accept method implementation in PetTree.h

An accept method is created in the PetTree class so that it will accept different kinds of visitors and perform different kinds of visit functions based on the visitor that is parsed in.

```
CharTree.accept(PreviewVisitor<string>());
CharTree.accept(DisplayVisitor<string>());
```

Figure 23: Visitor being used in the main file

The whole operation is carried out in the main file where the tree is created, and two visitors were used to demonstrate the different kinds of function that the visitor can do based on the object getting parsed in.

Reasoning

The reason behind using visitor pattern for my tree is because I wanted to include different operations for my tree but then I felt like it does not related to any of the classes. I felt like visitor is the most suitable in this case because it enables me to create methods that may be unrelated to the classes. I chose this pattern over others as I feel like this way of doing it is quite optimal in a sense, I do not have to touch the classes I made in order to change some operations to it. Other patterns do not provide such functionality and convenience at least to my knowledge.

Performance Justification

The visitor has a time complexity of O(n) which is mostly caused by the recursion that takes place within the accept() method inside of PetTree class. The time complexity might be decent, but the time taken may not be as good as it involves quite a bit of redirection from one class to another which may increase the time gap between each class traversal. This will not be the case if I were to implement the program without the visitor. But doing so I would sacrifice so much more just to save a little time. Even then, the difference between having it and not having it was not very noticeable in terms of memory space consumption as it does not consume much memory to process or run the code. So, I would rather implement visitor just for the sake of gaining all these benefits of convenience, efficiency and decent performance that comes with little to no cost on the performance side of the system.

Troubleshooting

I faced quite a few issues when it comes to understanding this pattern. I had a hard time grasping the concept of visitors as it feels like the execution of this whole thing to be very complicated. I was very confused on how the entire thing is connected as it involves quite a few classes and I had a hard time trying to identify what is considered a visitor and what is not. Fortunately, the tutorial classes did provide me with some form of referencing where I can understand, on a basic level, on how to implement it. I also had a lot of help from online resources such as the Youtube videos teaching me the concept of visitors (Banas 2012). I also had some online references from articles that gave me examples of how to implement this pattern into my program (Visitor in C++: Before and after n.d.).

9. Other Design Pattern: Singleton

Description

The singleton design pattern is a design pattern that only allows for one instance of the class to exist. This means that the class cannot be called multiple times like other object classes. This is a very useful pattern to utilize given that the situation calls for only needing one instance of the object to coordinate the entire program. The object usually consist of crucial information that the program requires to run but the information may need to be used multiple times and creating multiple instances of the same object may cause some memory space issues. An essential part of a singleton class is to have its constructor hidden in the private field to avoid getting called and the only way to access it is to use a get function that gets the entire class.

Purpose

```
class Battledex
{//this class is essentially used to cre
    //this class includes a few concepts
    //this class uses the singleton desi

public:
    static Battledex& Get()
    {
        static Battledex PokeInstance;
        return PokeInstance;
    }//this gets the instance in the cla
    Battledex(const Battledex&) = delete;

private:
    Battledex();//constructor is placed in
```

Figure 24: Singleton implementation in Battledex Class

Singleton is applied in the Battledex class. This is mainly because most of the data inside of this class is required throughout the program in order to run successfully. I mostly use the data from the Battledex class in my main file where I need to create a number of different pets and other data.

```
Battledex& Battledex = Battledex::Get();/
```

Figure 25: Get call inside of main file

The class is called using the get function which is Pokedex::Get() and a pointer is applied for the purpose of ease of use.

Reasoning

The main reason why I chose to use singleton is because of the sheer number of data that the Battledex may have. I want to ensure that the program can run in an optimal condition and constantly creating the same class to get the information will hinder that as every class instantiated will consume some memory space which will in turn increase complexity of the program. Due to the class having quite a number of data inside, I realized I need singleton so that I only need to create one instance as I may need to keep using the data from there throughout the program which will in turn result in consuming a lot more memory than necessary. This was in turn successful as I used the data from that class multiple times to create the player's pets, the enemy's pets and many other all while only needing to create one instance of the class.

Performance Justification

The big-o notation suitable for this scenario is O(1). Since I already explained that singleton has only one instance, it would make sense that the time complexity is O(1) as it only executes once with constant time. I feel like this design pattern performs extremely well with O(1) and it only needing one instance means that the memory it takes would be very low as well. This is by far one of the best design patterns for improving performance in terms of complexity and system memory consumption inside my program. One downside is that using this design pattern is highly situational where only when the same data is needed multiple times does singleton be effective.

Troubleshooting

I did not face much of a hurdle this time as I already had the experience of implementing this design pattern from my programming project 1 which also gave me the reference to recreate this design pattern again.

10. Application Programming Interface (API)

Description

An application programming interface (API) is a computing interface where it interacts with at least two different software intermediaries. Basically, APIs serves as a link between two systems and provide some form of interaction between the two systems to either make requests, or to collect data from one of the systems and using it on the other. APIs also act as a bridge to further extend a certain program's functionality by combining features of two systems together. A benefit of using API is that it does not require the developer to understand the entire library or system to use it as the developer can just use the specific part that they need.

Purpose

The API that I chose to use is a web API where it connects to an existing app online. I chose to use an API which connects me to a famous social media app called Twitter. The library that I chose to use is Twitcurl. Twitcurl is a C++ library that uses cURL to handle requests and responses between my program and Twitter. This library will help me connect to Twitter REST APIs and essential run some commands to carry out some basic Twitter features such as making tweets.

The library is downloaded from a GitHub site which also gives some documentation on what it is and how to use it (swatkat 2017). After that, I included the files into my current solution and began the process of linking the library with my program.

```
#include "..\..\twitcurl\libtwitcurl\oauthlib.h"
#include "..\..\twitcurl\libtwitcurl\twitcurl.h"

FILE _iob[] = { *stdin, *stdout, *stderr };
extern "C" FILE * __cdecl __iob_func(void) { return _iob; }
extern "C" void impset_output_format(void) {};
```

Figure 26: Preparing twitcurl and connections

After the linking process, all I need to do is to include the library in the file that I am going to use it on, in my case is the main file.

```
twitCurl twitter;

//keys that are required to access twitter services and API
twitter.getOAuth().setConsumerKey("fGRwDRVgQoM3CDesgebuMn9sN");
twitter.getOAuth().setConsumerSecret("LrvyxMsHa7PPnvTWhgzdfxzWbpj9RfI7oKZ1TW9cgaoluBPjsR");
twitter.getOAuth().setOAuthTokenKey("1088469456429273088-n1TrkK59SzXm6TccU0wEBsQywP3NvN");
twitter.getOAuth().setOAuthTokenSecret("LTRsPQeDazQ93XRimycdaJQX0kTz0ZYEadS1c8yHhPwCQ");
```

Figure 27: Accessing Twitter with token keys given

```
twitter.statusUpdate(tweet);
```

Figure 28: Using twitcurl to make a tweet

In order for me to use Twitter from my end of the program, I need to gain permission from Twitter, and I did so through Twitter developer which is where I would get my access token key to use Twitter from my program. After that, all I need to do is to create the twitCurl object and use the methods inside of it to carry out my features which is to make a tweet.

Reasoning

The reason behind why I chose this API over others is simple. I just wanted a web API that connects me to some sort of social media just so I can make status updates to show off my game. At the moment, I did find quite a few APIs but they either do no support C++ implementation or they are lacking of some documentation which made it hard for me to implement it. Twitcurl seems to me to be the best choice at the moment as it is a C++ library and it also has an understandable documentation with some Youtube videos guiding me as well.

Performance Justification

In this program, I only created the library object once and connect it to Twitter's API. This would mean that the time complexity is O(1) as it only has one execution. Even so, I think that the memory consumption might increase drastically as the program has to be linked into another library which contains a lot more methods and elements inside of it. The fact that the library I used is the one that connects to the internet may hinder the performance as well. So, in terms of time complexity, it is very good but in terms of memory, it may come to be at a disadvantage. Even so, I feel like not implementing this would be a lost cause because of the features that it brings, the fact that it enables me to use another entirely different application is worth the sacrifice for some memory space. Since the API can be done by abstracting where the developer need only the methods that they need, the memory consumption may not pose so much of an obstacle given if their situation is like me where I only require the API to make tweets.

Troubleshooting

The process of implementing the API was a very painful one to say the least. This is mostly due to my lack of knowledge of what is an API to begin with paired up with the topic not being taught during lectures which made me research all these from scratch. I faced many issues when dealing with this implementation. Firstly, the documentation did confuse me quite a bit because of how some instructions either not being clear or not very descriptive. Secondly, I had almost no previous knowledge of how to link libraries and APIs. These lead me into a lot of errors that I need to fix and research on. Fortunately, I had friends who are attempting similar APIs so I can go to them for help and deal with this together.

A few examples of issues I faced are as below:

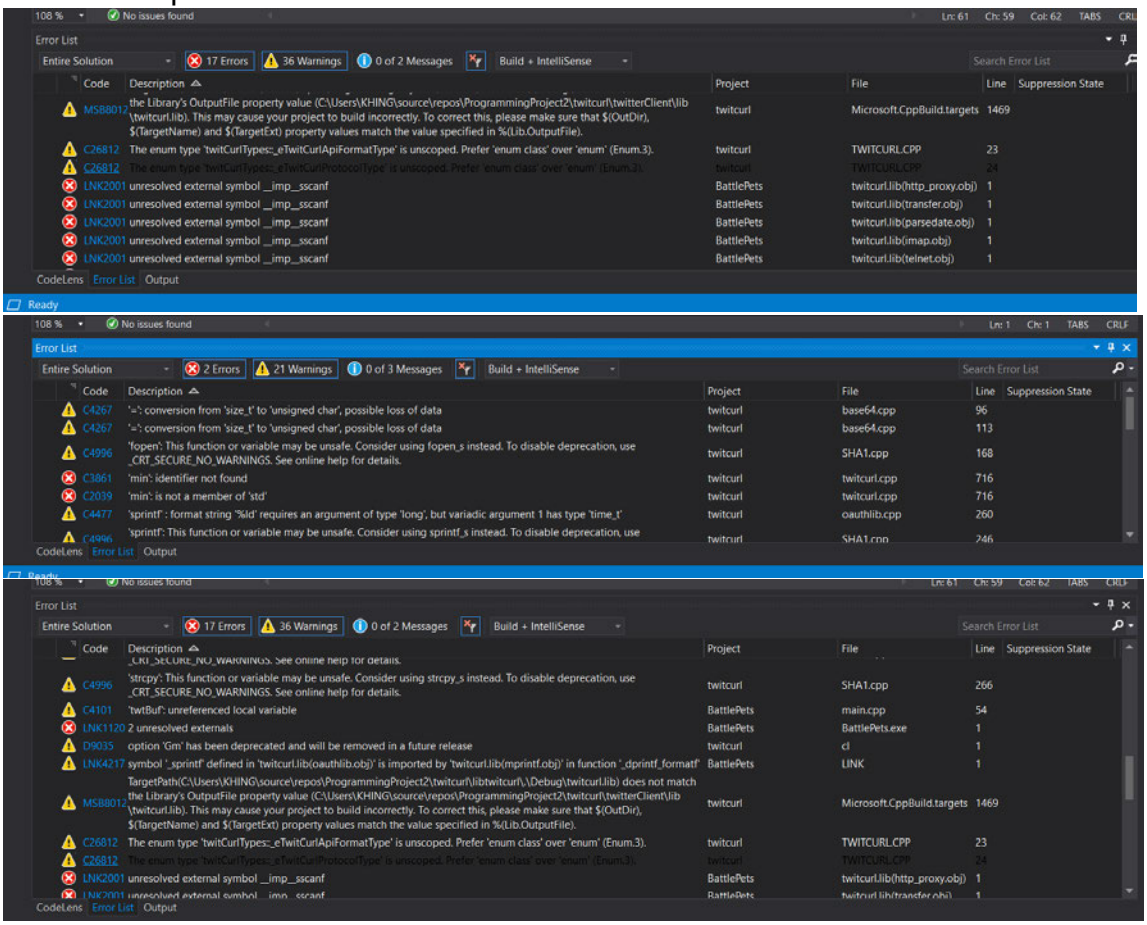

Figure 29: Linking errors i faced

As seen from the above screenshots, most of it are related to linking issues. In order to resolve this issue, I had to meticulously ensure that both my program and the library are linked properly, and no mismatches were found. Only when all of the mismatches are resolved the program could build successfully.

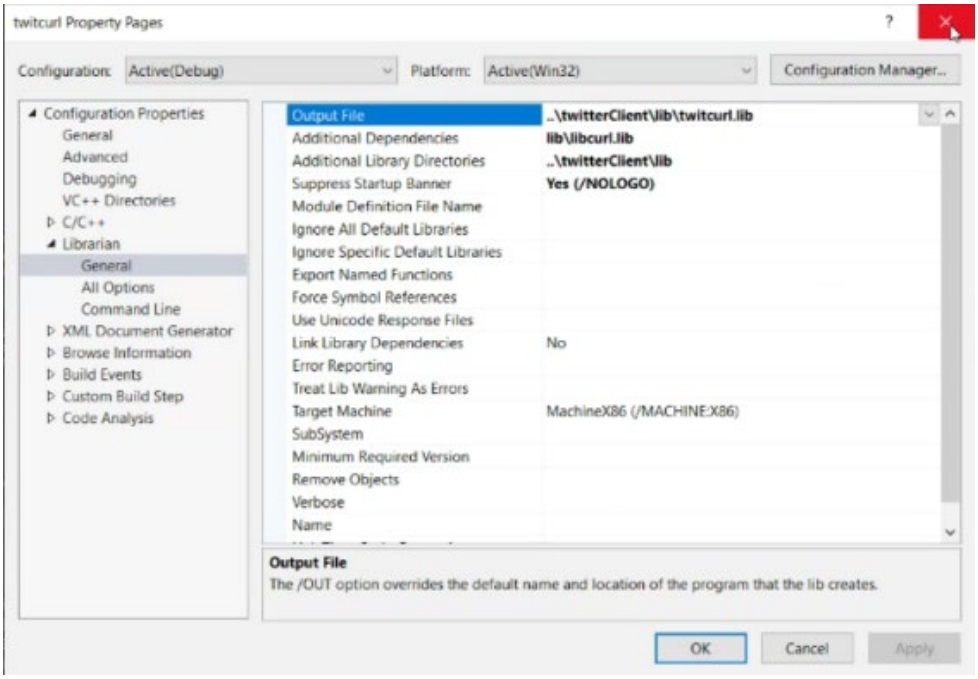

Figure 30: Changing the properties in twitcurl

The above screenshot shows me changing the output file field into the path of the library and then adding some dependencies to the library.

I also faced some issues with the coding part where it shows an error such as the screenshot below.

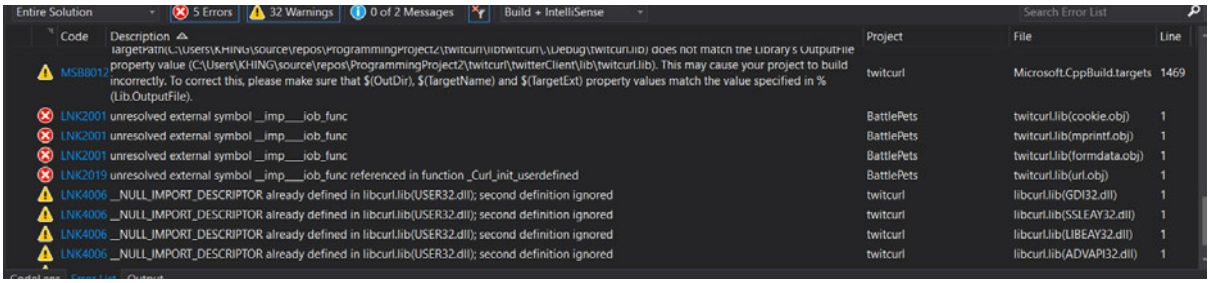

Figure 31: Error message of a mistyped code

I did not understand what the errors was about so I searched online for the resolution and it turned out that the extern "C" FILE * _cdecl _iob_func(void) { return _iob; } was not written properly as it was lacking additional underscores. The correct way of writing it is extern "C" FILE * __cdecl __iob_func(void) { return _iob; } (Fred 2015).

4 References

Adamchik, VS 2009, Binary Trees, CMU, viewed 28 May 2020, <https://www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/trees.html#:~:text=The%20recursive%20structure%20of%20a,to%20find%20a%20particular%20node.>.

Banas, D 2012, Visitor Design Pattern, 2 November, viewed 26 May 2020, <https://www.youtube.com/watch?v=pL4mOUDi54o&t=334s>.

dwightcOSU 2015, Pokemon Battle Game, GitHub, viewed 27 April 2020, < https://github.com/dwightcOSU/PokemonBattleGame>.

Fred 2015, unresolved external symbol __imp____iob_func referenced in function _OpenSSLDie, StackOverflow, viewed 29 May 2020, <https://stackoverflow.com/questions/30450042/unresolved-external-symbol-imp-iob-func-referenced-in-function-openssldie/35339896>.

Swatkat 2017, Twitcurl, GitHub, viewed 27 May 2020, <https://github.com/swatkat/twitcurl>.

Queue implementation using templates in C++, n.d., Techie Delight, viewed 22 May 2020, <https://www.techiedelight.com/queue-implementation-using-templates-cpp/>.

Visitor in C++: Before and after, n.d., Source Making, viewed 25 May 2020, <https://sourcemaking.com/design_patterns/visitor/cpp/1>.

5 Appendix

Main.cpp

```
using namespace std;
#include <iostream>
#include "Moves.h"
#include "Battledex.h"
#include "Owner.h"
#include <windows.h>
#include <string>
#include "PetStack.h"
#include "Moves.h"
#include "PetTree.h"
#include "PetVisitor.h"
#include <algorithm>
#include <cctype>
//including twitcurl libraries for the API
#include "..\..\twitcurl\libtwitcurl\oauthlib.h"
#include "..\..\twitcurl\libtwitcurl\twitcurl.h"

FILE _iob[] = { *stdin, *stdout, *stderr };
extern "C" FILE * __cdecl __iob_func(void) { return _iob; }
extern "C" void impset_output_format(void) {};

//a simple function to remove whitespace for the
string removeWhiteSpace(string word)
{
	string temp = word;
	temp.erase(remove(temp.begin(), temp.end(), '\n'), temp.end());
	return temp;
}

void drawBanner()
{//this is the banner for aesthetic purposes
	cout << R"(
 _______  _______  _______  _______  ___      _______  _______  _______  _______
_______     |
|  _    ||   _   ||       ||       ||   |    |       ||       ||       ||       ||
|   |
| |_|   ||  |_|  ||_     _||_     _||   |    |    ___||    _  ||    ___||_     _||
_____|   |
|       ||       |  |   |    |   |  |   |    |   |___ |   |_| ||   |___   |   |  |
|_____     |
|  _   | |       |  |   |    |   |  |   |___ |    ___||    ___||    ___|  |   |
|_____  |   |
| |_|   ||   _   |  |   |    |   |  |       ||   |___ |   |    |   |___   |   |
_____| |   |
|_______||__| |__|  |___|    |___|  |_______||_______||___|    |_______|  |___|
|_______|   |
_____________________________________________________________________________
_______|)" << endl;
}

int main()
{
	system("CLS");

	//create the twiCurl object so that i can link it to my twitter account
	twitCurl twitter;

	//keys that are required to access twitter services and API
```

```
	twitter.getOAuth().setConsumerKey("fGRwDRVgQoM3CDesgebuMn9sN");
	twitter.getOAuth().setConsumerSecret("LrvyxMsHa7PPnvTWhgzdfxzWbpj9RfI7oKZ1TW9cg
aoluBPjsR");
	twitter.getOAuth().setOAuthTokenKey("1088469456429273088-
n1TrkK59SzXm6TccU0wEBsQywP3NvN");
	twitter.getOAuth().setOAuthTokenSecret("LTRsPQeDazQ93XRimycdaJQX0kTz0ZYEadS1c8y
HhPwCQ");
	drawBanner();

	string welcome = "\nHello and welcome to Battle Pets!!\nHere you will
experience a short snippet of text-based fighting with various pets of ur own!!\nThis
game has three types of pets, Fire, Water and Grass.\nYou are to fight a computer
opponent using a team of pets. \nBut before we start, you get to choose your
character!!\nPress Enter to proceed!!";
	int x = 0;
	while (welcome[x] != '\0')
	{
		cout << welcome[x];
		Sleep(30);
		x++;
	};
	cin.get();//makes a small animation-like output for aesthetic purposes


	//Tree
	//the nodes are read from the text file
	vector<string> treeString;
	string lines;
	ifstream textfile("TreeStrings.txt");
	if (textfile.is_open())
	{
		while (getline(textfile, lines))
		{
			treeString.push_back(lines);
		}
		textfile.close();
	}
	//some formatting on some strigs
	string newlinestring1 = "\n" + treeString[1];
	string newlinestring2 = "\n" + treeString[2];
	string newlinestring3 = "\n" + treeString[3];
	string newlinestring4 = "\n" + treeString[5];

	//turning the strings into tree nodes
	PetTree<string> CharTree(treeString[0]);
	PetTree<string> TreeNode1(newlinestring1);
	PetTree<string> TreeNode2(newlinestring2);
	PetTree<string> TreeNode3(newlinestring3);
	PetTree<string> TreeNode4(treeString[4]);
	PetTree<string> TreeNode5(newlinestring4);
	PetTree<string> TreeNode6(treeString[6]);

	//nodes on the 1st layer
	CharTree.attachLeft(&TreeNode1);
	CharTree.attachRight(&TreeNode2);

	//nodes on the 2nd layer
	CharTree.left().attachLeft(&TreeNode3);//Leo, Mew, Aqua
	CharTree.left().attachRight(&TreeNode4);//Leon, Mewmew, Aquality

	//nodes on the 2nd layer
	CharTree.right().attachLeft(&TreeNode5);//Leoreon, Mewmian, Aquanox
```

```
	CharTree.right().attachRight(&TreeNode6);//Fierry, Leafy, Splashy

	Battledex& Battledex = Battledex::Get();//declare a pointer to the singleton to get the Pets data later on
	PetStack<Pets> Pets_Stack(3);//initialize stack

	cout << "Here is a preview of characters you can choose along with their pets" << endl;
	CharTree.accept(PreviewVisitor<string>());


	//the text file needs to be removed everytime before running to avoid writing duplicate nodes into it
	//this is caused from the recurssion inside the accept method where each key is parsed into the visit function which made me unable to write it all at once
	remove("BattlePetsData.txt");
	CharTree.accept(DisplayVisitor<string>());

	Pets team[3];//an array that is used to store the player's team before the pets get enqueued into the queue
	Owner owner;

	//display character options from the tree and ask for player input
	cout << "\nChoose your character! Each character have a different set of pokemon so choose wisely\nPlease input number (1/2):" << endl;
	cout << "[1]" << removeWhiteSpace(CharTree.left().key()) << "\n or \n[2]" << removeWhiteSpace(CharTree.right().key()) << endl;
	int character;
	cin >> character;
	while (true)
	{
		if (character == 1)
		{
			owner.setName("Ash");//creating the array class
			int teamChoice;
			//display team options from the tree and ask for player input
			cout << "Choose your pet team" << endl;
			cout << "[1]" << removeWhiteSpace(CharTree.left().left().key()) << " or [2]" << CharTree.left().right().key() << endl;
			cout << "Enter the number (1/2): ";
			cin >> teamChoice;
			while (true)
			{
				if (teamChoice == 1)
				{
					team[0] = Battledex.getFirePet("Leo").getPet();
					team[1] = Battledex.getGrassPet("Mew").getPet();
					team[2] = Battledex.getWaterPet("Aqua").getPet();
					break;
				}
				else if (teamChoice == 2)
				{
					team[0] = Battledex.getFirePet("Leon").getPet();
					team[1] = Battledex.getGrassPet("Mewmew").getPet();
					team[2] = Battledex.getWaterPet("Aquality").getPet();
					break;
				}
				else
				{
					cout << "Wrong input please try again" << endl;
					cout << "Choose your pet team" << endl;
```

```
                                cout << "[1]" << CharTree.left().left().key() << "
or [2]" << CharTree.left().right().key() << endl;
                                cout << "Enter the number (1/2): ";
                                cin >> teamChoice;
                                continue;
                        }
                }
                break;

        }
        else if (character == 2)
        {
                owner.setName("Amy");//creating the array class
                int teamChoice;
                //display team options from the tree and ask for player input
                cout << "Choose your pet team" << endl;
                cout << "[1]" << removeWhiteSpace(CharTree.right().left().key())
<< " or [2]" << CharTree.right().right().key() << endl;
                cout << "Enter the number (1/2): ";
                cin >> teamChoice;
                while (true)
                {
                        if (teamChoice == 1)
                        {
                                team[0] = Battledex.getFirePet("Leoreon").getPet();
                                team[1] =
Battledex.getGrassPet("Mewmian").getPet();
                                team[2] =
Battledex.getWaterPet("Aquanox").getPet();
                                break;
                        }
                        else if (teamChoice == 2)
                        {
                                team[0] = Battledex.getFirePet("Fierry").getPet();
                                team[1] = Battledex.getGrassPet("Leafy").getPet();
                                team[2] =
Battledex.getWaterPet("Splashy").getPet();
                                break;
                        }
                        else
                        {
                                cout << "Wrong input please try again" << endl;
                                cout << "Choose your pet team" << endl;
                                cout << "[1]" << CharTree.right().left().key() << "
or [2]" << CharTree.right().right().key() << endl;
                                cout << "Enter the number (1/2): ";
                                cin >> teamChoice;
                                continue;
                        }
                }
                break;

        }
        else
        {
                cout << "Wrong input please try again" << endl;
                cout << "Choose your character! Each character have a different
set of pokemon so choose wisely\n Please input number (1/2):";
                cin >> character;
                continue;
        }
    }
```

```
	//destroy the tree as it no longer is in use
	CharTree.left().detachLeft();
	CharTree.left().detachRight();
	CharTree.right().detachLeft();
	CharTree.right().detachRight();
	CharTree.detachLeft();
	CharTree.detachRight();


	//prints out the Pets in the array
	cout << "Hello "<< owner.getName() <<"!\nHere are your pets: " << endl;
	for (int i = 0; i < 3; i++)
	{
		cout << i + 1 << ". "<< team[i].getName() << endl;
	}
	string yn;
	//letting the player rearrange the pets if they wish to
	cout << "You will have to fight using this team in the exact order. Do you wish
to change the arrangement of your team?[Yes/No]" << endl;
	cin >> yn;

	if (yn == "Yes" || yn == "yes")
	{
		int new_position;
		Pets new_team[3];
		for (int i = 0; i < 3; i++) {
			cout << i + 1 << ". " << team[i].getName();
			cout << "\tNew position(input a number): ";
			cin >> new_position;
			new_team[new_position - 1] = team[i];
		}
		cout << "Updated team order: " << endl;
		for (int i = 0; i < 3; i++) {//adding the Pets from the above array into
the queue in the object class owner
			cout << i + 1 << ". " << new_team[i].getName() << endl;
			owner.enqueue(new_team[i]);
		}

	}
	else
	{
		for (int i = 0; i < 3; i++) {//adding the Pets from the above array into
the queue in the object class owner
			owner.enqueue(team[i]);

		}
	}

	//a small menu for the player
	int player_choice;
	cout << "Do you wish to: \n[1] View Battledex \n[2] Start Fight \n[0] Exit
Program" << endl;
	cin >> player_choice;
	system("CLS");//this is used to clear the screen to make it look more neat
	while (player_choice != 0)
	{
		if (player_choice == 1)//if the player selects to view the Battledex
		{
			int Pets_type;
			drawBanner();
			cout << "Select what kind of Pets you want to see: " << endl;
```

```
                    cout << "[1] Water \t[2]Fire \t[3]Grass \t[4]View Pets Moves
\t[0]Go Back" << endl;
                    cout << "Enter a number (1/2/3/0): ";
                    cin >> Pets_type;
                    system("CLS");

                    if (Pets_type == 1)//water Battledex
                    {
                           drawBanner();
                           //prints a preview of the Pets available in the water
category
                           Battledex.printWaterPet();
                           string poke_name;
                           cout << "Enter the Pets name EXACTLY for details (Case
Sensitive)" << endl;
                           cin >> poke_name;

                           WaterPet results =
Battledex.getWaterPet(poke_name).getPet();
                           cout << "ID: " << results.getID() << "\tName: " <<
results.getName() << "\tHP: " << results.getMaxHP() << "\tMoves: " <<
results.getMove(1).getName() << ", " << results.getMove(2).getName() << ", " <<
results.getMove(3).getName() << ", " << results.getMove(4).getName();
                           results.printType();//using the override method to print
the types

                           bool iterating = true;
                           //here shows the use of doubly linked list where the
player can iterate forwards and backwards through each Pets
                           while (iterating == true)
                           {
                                  string iterate_input;
                                  cout << "[,] Previous \t [.] Next \t [/] Exit" <<
endl;
                                  cin >> iterate_input;
                                  if (iterate_input == ",")//get the previous Pets in
the Battledex
                                  {
                                         WaterPet results =
Battledex.PreviousWaterPet(poke_name).getPet();
                                         cout << "ID: " << results.getID() << "\tName:
" << results.getName() << "\tHP: " << results.getMaxHP() << "\tMoves: " <<
results.getMove(1).getName() << "," << results.getMove(2).getName() << "," <<
results.getMove(3).getName() << "," << results.getMove(4).getName();
                                         results.printType();
                                         poke_name = results.getName();
                                         iterating = true;
                                  }
                                  else if (iterate_input == ".")//gets the next Pets
in the Battledex
                                  {
                                         WaterPet results =
Battledex.NextWaterPet(poke_name).getPet();
                                         cout << "ID: " << results.getID() << "\tName:
" << results.getName() << "\tHP: " << results.getMaxHP() << "\tMoves: " <<
results.getMove(1).getName() << "," << results.getMove(2).getName() << "," <<
results.getMove(3).getName() << "," << results.getMove(4).getName();
                                         results.printType();
                                         iterating = true;
                                         poke_name = results.getName();
                                  }
                                  else if (iterate_input == "/")
```

```
                                        {
                                                system("CLS");
                                                break;
                                        }
                                        else
                                        {
                                                cout << "Invalid Output. Please try again."
<< endl;
                                                continue;
                                        }
                                }

                        }
                        else if (Pets_type == 2)//fire Battledex
                        {
                                drawBanner();
                                Battledex.printFirePet();
                                string poke_name;
                                cout << "Enter the Pets name EXACTLY for details (Case
Sensitive)" << endl;
                                cin >> poke_name;
                                FirePet results =
Battledex.getFirePet(poke_name).getPet();
                                cout << "ID: " << results.getID() << "\tName: " <<
results.getName() << "\tHP: " << results.getMaxHP() << "\tMoves: " <<
results.getMove(1).getName() << ", " << results.getMove(2).getName() << ", " <<
results.getMove(3).getName() << ", " << results.getMove(4).getName();
                                results.printType();

                                bool iterating = true;
                                //use of doubly linked list
                                while (iterating == true)
                                {
                                        string iterate_input;
                                        cout << "[,] Previous \t [.] Next \t [/] Exit" <<
endl;
                                        cin >> iterate_input;
                                        if (iterate_input == ",")
                                        {
                                                FirePet results =
Battledex.PreviousFirePet(poke_name).getPet();
                                                cout << "ID: " << results.getID() << "\tName:
" << results.getName() << "\tHP: " << results.getMaxHP() << "\tMoves: " <<
results.getMove(1).getName() << "," << results.getMove(2).getName() << "," <<
results.getMove(3).getName() << "," << results.getMove(4).getName();
                                                results.printType();
                                                poke_name = results.getName();
                                                iterating = true;
                                        }
                                        else if (iterate_input == ".")
                                        {
                                                FirePet results =
Battledex.NextFirePet(poke_name).getPet();
                                                cout << "ID: " << results.getID() << "\tName:
" << results.getName() << "\tHP: " << results.getMaxHP() << "\tMoves: " <<
results.getMove(1).getName() << "," << results.getMove(2).getName() << "," <<
results.getMove(3).getName() << "," << results.getMove(4).getName();
                                                results.printType();
                                                iterating = true;
                                                poke_name = results.getName();
                                        }
                                        else if (iterate_input == "/")
```

```
                                        {
                                                system("CLS");
                                                break;
                                        }
                                        else
                                        {
                                                cout << "Invalid Output. Please try again."
<< endl;
                                                continue;
                                        }
                                }
                        }
                        else if (Pets_type == 3)//grass Battledex
                        {
                                drawBanner();
                                Battledex.printGrassPet();
                                string poke_name;
                                cout << "Enter the Pets name EXACTLY for details (Case
Sensitive)" << endl;
                                cin >> poke_name;
                                GrassPet results =
Battledex.getGrassPet(poke_name).getPet();
                                cout << "ID: " << results.getID() << "\tName: " <<
results.getName() << "\tHP: " << results.getMaxHP() << "\tMoves: " <<
results.getMove(1).getName() << ", " << results.getMove(2).getName() << ", " <<
results.getMove(3).getName() << ", " << results.getMove(4).getName();
                                results.printType();
                                bool iterating = true;
                                //use of doubly linked list
                                while (iterating == true)
                                {
                                        string iterate_input;
                                        cout << "[,] Previous \t [.] Next \t [/] Exit" <<
endl;
                                        cin >> iterate_input;
                                        if (iterate_input == ",")
                                        {
                                                GrassPet results =
Battledex.PreviousGrassPet(poke_name).getPet();
                                                cout << "ID: " << results.getID() << "\tName:
" << results.getName() << "\tHP: " << results.getMaxHP() << "\tMoves: " <<
results.getMove(1).getName() << "," << results.getMove(2).getName() << "," <<
results.getMove(3).getName() << "," << results.getMove(4).getName();
                                                results.printType();
                                                poke_name = results.getName();
                                                iterating = true;
                                        }
                                        else if (iterate_input == ".")
                                        {
                                                GrassPet results =
Battledex.NextGrassPet(poke_name).getPet();
                                                cout << "ID: " << results.getID() << "\tName:
" << results.getName() << "\tHP: " << results.getMaxHP() << "\tMoves: " <<
results.getMove(1).getName() << "," << results.getMove(2).getName() << "," <<
results.getMove(3).getName() << "," << results.getMove(4).getName();
                                                results.printType();
                                                iterating = true;
                                                poke_name = results.getName();
                                        }
                                        else if (iterate_input == "/")
                                        {
                                                system("CLS");
```

```
                                        break;
                                }
                                else
                                {
                                        cout << "Invalid Output. Please try again."
<< endl;
                                        continue;
                                }
                        }
                }
                else if (Pets_type == 4)//display all the available moves in the
singly linked list
                {
                        drawBanner();
                        Battledex.getMoves();
                        system("PAUSE");
                        system("CLS");
                        drawBanner();
                        cout << "Do you wish to: \n[1] View Battledex \n[2] Start
Fight \n[0] Exit Program" << endl;
                        cin >> player_choice;
                        continue;
                }

else if (Pets_type == 0)
                {
                        system("CLS");
                        drawBanner();
                        cout << "Do you wish to: \n[1] View Battledex \n[2] Start
Fight \n[0] Exit Program" << endl;
                        cin >> player_choice;
                        continue;
                }
        }

        //if the player chooses to battle
        if (player_choice == 2)
        {
                system("CLS");
                drawBanner();
                string opponent = "Looks like your opponent is approaching.\nYou
will be going up against one of the water Pets masters. \nPets master: Water you
doing?? Get it? Get it? Haha.\nINSTRUCTIONS:\nYou can fight by inputting the move
number when it appears on the screen.\nBe careful of your HP shown next to your Pets's
name.\nOk, Ready??\nPress Enter if you are ready\n";
                cin.get();
                system("CLS");
                drawBanner();
                int y = 0;
                while (opponent[y] != '\0')
                {
                        cout << opponent[y];
                        Sleep(50);
                        y++;
                };
                //animation-style output for better appearance

                //creates an array of enemy Pets the player is going to face
                Pets enemy_team[] = { Battledex.getWaterPet("Starking").getPet(),
Battledex.getWaterPet("Aqua").getPet(), Battledex.getWaterPet("Aquality").getPet()};

                //pushes the enemy into the stack to be faced in an order
```

```
			for (int i = 0; i < 3; i++) {
				Pets_Stack.push(enemy_team[i]);
			}


			//fighting commence
			cout << "Pets Master: I choose you, " <<
Pets_Stack.top().getName() << "!!!" << endl;
			int x = owner.getTeamSize();//this will be used to see which Pets
team is completely depleted and decide the winner

			while (!Pets_Stack.isEmpty() && x > 0)
			{
				//creates some indication of current enemy Pets and your
own Pets
				cout << "Enemy Pets - " << Pets_Stack.top().getName() <<
"\tHP: " << Pets_Stack.top().getCurrentHP() << "/" << Pets_Stack.top().getMaxHP() <<
endl;

				cout << "-----------------------------------------"
<< endl;
				cout << "Your Pets - " << owner.peek().getName() << "\tHP:
" << owner.peek().getCurrentHP() << "/" << owner.peek().getMaxHP() << endl;
				cout << "Pick a move: " << endl;

				//displays the moves for the Pets
				int move_no;
				for (int j = 1; j <= 4; j++)
				{
					cout << "[" << j << "] " <<
owner.peek().getMove(j).getName() << endl;
				}
				cout << "Enter your move (1/2/3/4): ";
				cin >> move_no;
				system("CLS");
				drawBanner();
				while (move_no > 4)
				{
					cout << "Wrong input please try again." << endl;
					cout << "Enter your move (1/2/3/4): ";
					cin >> move_no;
				}
				//tells the player what move is used and decrease the hp
of the enemy Pets
				cout << owner.peek().getName() << " used " <<
owner.peek().getMove(move_no).getName() << endl;

	Pets_Stack.top().setHP(owner.peek().getMove(move_no).getdmg());


				//if the enemy Pets health is lesser or equal to zero
				if (Pets_Stack.top().getCurrentHP() <= 0)
				{
					cout << Pets_Stack.top().getName() << " has
fainted!" << endl;
					Pets_Stack.pop();//the Pets will be removed from
the stack
					continue;
				}
				else
				{
```

```
                                //if the Pets has not fainted, then the enemy Pets
will randomly choose a move and attack the player
                                int random_move = rand() % 4 + 1;
                                cout << "Enemy " << Pets_Stack.top().getName() << "
used " << Pets_Stack.top().getMove(random_move).getName() << endl;

      owner.peek().setHP(Pets_Stack.top().getMove(random_move).getdmg());//decrease
the player Pets's hp

                                if (owner.peek().getCurrentHP() <= 0)//if the
player's Pets dies
                                {
                                        cout << "Your Pets - " <<
owner.peek().getName() << " has fainted" << endl;
                                        owner.dequeue();//removed from the list
                                        x--;
                                        unsigned int input;
                                        cout << "Do you want to: \n[1] Switch to the
next Pet \n[2] Exit Program" << endl;
                                        cout << "Enter your choice:";
                                        cin >> input;
                                        system("CLS");
                                        drawBanner();
                                        while (input > 2)
                                        {
                                                cout << "Invalid choice" << endl;
                                                cout << "Do you want to: \n[1] Switch
to the next Pets \n[2] Exit Program" << endl;
                                                cout << "Enter your choice:";
                                                cin >> input;
                                        }
                                        if (input == 1) {//if the player chooses to
switch Pets, it will move to the next Pets in the array
                                                continue;
                                        }
                                        else if (input == 2)
                                        {//if not the program will end
                                                return 0;
                                        }

                                }
                        }
                }
                if (Pets_Stack.isEmpty())//if enemy stack is empty
                {
                        cout << "CONGRATULATIONS YOU WIN!!" << endl;
                        char tweeting;
                        //prompting the player to make the tweet
                        cout << "Like my game? SHARE IT TO THE WORLD!![Y/N]" <<
endl;
                        cin >> tweeting;

                        while (true)
                        {
                                if (tolower(tweeting) == 'y')
                                {
                                        //message that will appear on twitter
                                        string tweet = "I just finished my game of
BattlePets as " + owner.getName() + ", it is a great text-based action game come and
join me on the fun";
                                        cout << "This Tweet will be posted on our
page on Twitter: " << endl;
```

```
                                        cout << tweet << endl;
                                        int tweetConfirm;
                                        cout << "[1]Confirm \t [2]Cancel" << endl;
                                        cin >> tweetConfirm;
                                        //if confirmed, make the tweet
                                        if (tweetConfirm == 1)
                                        {
                                                twitter.statusUpdate(tweet);
                                                cout << "Tweeted successfully! Thank
you for playing my game!!" << endl;
                                                return 0;
                                        }
                                        else if (tweetConfirm == 2)
                                        {
                                                return 0;
                                        }

                                }
                                else if (tolower(tweeting) == 'n')
                                {
                                        return 0;
                                }
                                else
                                {
                                        cout << "Invalid input please try again." <<
endl;
                                        cout << "Like my game? SHARE IT TO THE
WORLD!![Y/N]" << endl;
                                        cin >> tweeting;
                                        continue;

                                }
                        }


                }

                if (x <= 0)//if Pets array count is empty
                {
                        cout << "That was your last Pets, You Lost. Better Luck
Next Time.^('o')^" << endl;
                        char tweeting;
                        //prompting player to tweet
                        cout << "Like my game? SHARE IT TO THE WORLD!![Y/N]" <<
endl;
                        cin >> tweeting;
                        while (true)
                        {
                                if (tolower(tweeting) == 'y')
                                {
                                        //message that will appear on twitter
                                        string tweet = "I just finished my game of
BattlePets as " + owner.getName() + ", it is a great text-based action game come and
join me on the fun";
                                        cout << "This Tweet will be posted on our
page on Twitter: " << endl;
                                        cout << tweet << endl;
                                        int tweetConfirm;
                                        cout << "[1]Confirm \t [2]Cancel" << endl;
                                        cin >> tweetConfirm;
                                        //if confirmed, then tweet
                                        if (tweetConfirm == 1)
```

```
                                        {
                                                twitter.statusUpdate(tweet);
                                                cout << "Tweeted successfully! Thank
you for playing my game!!" << endl;
                                                return 0;
                                        }
                                        else if (tweetConfirm == 2)
                                        {
                                                return 0;
                                        }


                                }
                                else if (tolower(tweeting) == 'n')
                                {
                                        return 0;
                                }
                                else
                                {
                                        cout << "Invalid input please try again." <<
endl;
                                        cout << "Like my game? SHARE IT TO THE
WORLD!![Y/N]" << endl;
                                        cin >> tweeting;
                                        continue;
                                }
                        }

                }

        }
}

        return 0;
}
```

Chan Kwang Yung 101215067

Battledex.h

```
#pragma once
#include <vector>
#include <fstream>
#include "Pets.h"
#include "FirePet.h"
#include "WaterPet.h"
#include "GrassPet.h"
#include "BattledexList.h"
#include "Moves.h"
#include "MovesTemplate.h"
#include "MovesIterator.h"
using namespace std;

class Battledex
{//this class is essentially used to create all the data required to create the Pet and moves into a Battledex
	//this class includes a few concepts including singly linked list, doubly linked list, iterator, template class and singleton
	//this class uses the singleton design pattern to instantiate the data as only one instance is required for these data

public:
	static Battledex& Get()
	{
		static Battledex PokeInstance;
		return PokeInstance;
	}//this gets the instance in the class
	void printWaterPet();//prints all water Pet in the Battledex as a preview
	void printGrassPet();//prints all grass Pet in the Battledex as preview
	void printFirePet();//prints all fire Pet in the Battledex as preview

	//gets specific Pet from the doubly linked list based on the Pet name passed into the parameter
	//theres three getters for the three different  doubly linked list
	BattledexList<WaterPet>::Pets getWaterPet(string Pet);
	BattledexList<FirePet>::Pets getFirePet(string Pet);
	BattledexList<GrassPet>::Pets getGrassPet(string Pet);

	//iterates through water Pet doubly linked list
	BattledexList<WaterPet>::Pets PreviousWaterPet(string Pet);
	BattledexList<WaterPet>::Pets NextWaterPet(string Pet);

	//iterates through water Pet doubly linked list
	BattledexList<FirePet>::Pets PreviousFirePet(string Pet);
	BattledexList<FirePet>::Pets NextFirePet(string Pet);

	//iterates through water Pet doubly linked list
	BattledexList<GrassPet>::Pets PreviousGrassPet(string Pet);
	BattledexList<GrassPet>::Pets NextGrassPet(string Pet);

	void getMoves();

	Battledex(const Battledex&) = delete; //this prevents the creation of an object class that doesnt point to the instance

private:
	Battledex();//constructor is placed in private so that no one can initialize it from other classes

	//these are three different doubly linked list to store the three Pet types
```

```
        BattledexList<WaterPet> Water_Battledex;
        BattledexList<FirePet> Fire_Battledex;
        BattledexList<GrassPet> Grass_Battledex;


};
```

Chan Kwang Yung 101215067

Battledex.cpp

```
#include "Battledex.h"

//these global objects are created in order to be used by the below methods as many of
the methods require it to run properly
//these objects are the moves that are to be added into the singly linked list

Moves Tackle("Tackle", 10, "Normal");
Moves Cut("Cut", 15, "Normal");
Moves Water_whip("Water Whip", 20, "Water");
Moves Water_wave("Water Wave", 50, "Water");
Moves Vortex("Vortex", 65, "Water");
Moves Flaming_breath("Flaming Breath", 65, "Fire");
Moves Ember_bite("Ember Bite", 65, "Fire");
Moves Combustion("Combustion", 20, "Fire");
Moves Plant_protection("Plant protetion", 30, "Grass");
Moves Seed_gun("Seed gun", 50, "Grass");
Moves Flower_power("Flower power", 60, "Grass");
Moves Dance("Dance", 0, "Water");

//these objects are the various pokemon to be added into the doubly linked list
GrassPet Mew(1, "Mew", 100, 100, Plant_protection, Seed_gun, Flower_power, Tackle);
GrassPet Mewmew(2, "Mewmew", 100, 100, Plant_protection, Seed_gun, Flower_power,
Tackle);
GrassPet Mewmian(3, "Mewmian", 100, 100, Plant_protection, Seed_gun, Flower_power,
Tackle);
FirePet Leo(4, "Leo", 100, 100, Flaming_breath, Ember_bite, Combustion, Cut);
FirePet Leon(5, "Leon", 100, 100, Flaming_breath, Ember_bite, Combustion, Cut);
FirePet Leoreon(6, "Leoreon", 100, 100, Flaming_breath, Ember_bite, Combustion, Cut);
WaterPet Aqua(7, "Aqua", 100, 100, Water_whip, Water_wave, Vortex, Tackle);
WaterPet Aquality(8, "Aquality", 100, 100, Water_whip, Water_wave, Vortex, Tackle);
WaterPet Aquanox(9, "Aquanox", 100, 100, Water_whip, Water_wave, Vortex, Tackle);
WaterPet Starprince(129, "Starprince", 50, 50, Dance, Dance, Dance, Dance);
WaterPet Starking(130, "Starking", 100, 100, Water_whip, Water_wave, Vortex, Tackle);
WaterPet Splashy(134, "Splashy", 100, 100, Water_whip, Water_wave, Vortex, Tackle);
FirePet Fierry(136, "Fierry", 100, 100, Flaming_breath, Ember_bite, Combustion, Cut);
GrassPet Leafy(135, "Leafy", 100, 100, Plant_protection, Seed_gun, Flower_power,
Tackle);

//creating the nodes for each pokemon for the doubly linked list
BattledexList<WaterPet> Aqua_Battledex(Aqua);
BattledexList<WaterPet> Aquality_Battledex(Aquality);
BattledexList<WaterPet> Aquanox_Battledex(Aquanox);
BattledexList<WaterPet> Starprince_Battledex(Starprince);
BattledexList<WaterPet> Starking_Battledex(Starking);
BattledexList<WaterPet> Splashy_Battledex(Splashy);
BattledexList<FirePet> Leo_Battledex(Leo);
BattledexList<FirePet> Leon_Battledex(Leon);
BattledexList<FirePet> Leoreon_Battledex(Leoreon);
BattledexList<FirePet> Fierry_Battledex(Fierry);
BattledexList<GrassPet> Mew_Battledex(Mew);
BattledexList<GrassPet> Mewmew_Battledex(Mewmew);
BattledexList<GrassPet> Mewmian_Battledex(Mewmian);
BattledexList<GrassPet> Leafy_Battledex(Leafy);


Battledex::Battledex() //when this object gets instantiated for the 1st time, it
appends all the pokemon nodes
{
	Water_Battledex.append(Aqua_Battledex);
	Aqua_Battledex.append(Aquality_Battledex);
```

```
	Aquality_Battledex.append(Aquanox_Battledex);
	Aquanox_Battledex.append(Starprince_Battledex);
	Starprince_Battledex.append(Starking_Battledex);
	Starking_Battledex.append(Splashy_Battledex);

	Fire_Battledex.append(Leo_Battledex);
	Leo_Battledex.append(Leon_Battledex);
	Leon_Battledex.append(Leoreon_Battledex);
	Leoreon_Battledex.append(Fierry_Battledex);

	Grass_Battledex.append(Mew_Battledex);
	Mew_Battledex.append(Mewmew_Battledex);
	Mewmew_Battledex.append(Mewmian_Battledex);
	Mewmian_Battledex.append(Leafy_Battledex);

}

void Battledex::getMoves()
{//gets the moves based on the moves name
	typedef MovesNode<Moves> Move_Node;

	Move_Node move12(Dance);
	Move_Node move11(Flower_power, &move12);
	Move_Node move10(Seed_gun, &move11);
	Move_Node move9(Plant_protection, &move10);
	Move_Node move8(Combustion, &move9);
	Move_Node move7(Ember_bite, &move8);
	Move_Node move6(Flaming_breath, &move7);
	Move_Node move5(Vortex, &move6);
	Move_Node move4(Water_wave, &move5);
	Move_Node move3(Water_whip, &move4);
	Move_Node move2(Cut, &move3);
	Move_Node move1(Tackle, &move2);

	for (MovesIterator<Moves> iter(&move1); iter != iter.end(); ++iter)
	{
		cout << "Move Name: " << iter.getMoves().getName() << "\tDamage: " <<
iter.getMoves().getdmg() << "\tType: " << iter.getMoves().getType() << endl;
	}
}

BattledexList<WaterPet>::Pets Battledex::getWaterPet(string pokemon)
{
	typedef BattledexList<WaterPet>::Pets Battledex_water;
	//for loop is used to iterate through the doubly link list by using getNext
which iterators into the next node
	for (const Battledex_water* pn = &Water_Battledex.getNext(); pn !=
&Battledex_water::NIL; pn = &pn->getNext())
	{
		if (pn->getPet().getName() == pokemon)
		{
			return pn->getPet();
		}

	}
}


BattledexList<FirePet>::Pets Battledex::getFirePet(string pokemon)
{
	typedef BattledexList<FirePet>::Pets Battledex_fire;
```

```
	for (const Battledex_fire* pn = &Fire_Battledex.getNext(); pn !=
&Battledex_fire::NIL; pn = &pn->getNext())
	{
		if (pn->getPet().getName() == pokemon)
		{
			return pn->getPet();
		}

	}
}

BattledexList<GrassPet>::Pets Battledex::getGrassPet(string pokemon)
{
	typedef BattledexList<GrassPet>::Pets Battledex_grass;
	for (const Battledex_grass* pn = &Grass_Battledex.getNext(); pn !=
&Battledex_grass::NIL; pn = &pn->getNext())
	{
		if (pn->getPet().getName() == pokemon)
		{
			return pn->getPet();
		}
	}
}

BattledexList<WaterPet>::Pets Battledex::PreviousWaterPet(string pokemon)
{
	typedef BattledexList<WaterPet>::Pets Battledex_water;
	for (const Battledex_water* pn = &Water_Battledex.getNext(); pn !=
&Battledex_water::NIL; pn = &pn->getNext())
	{
		if (pn->getPet().getName() == pokemon)
		{
			return pn->getPrevious().getPet();
		}
	}
}

BattledexList<WaterPet>::Pets Battledex::NextWaterPet(string pokemon)
{
	typedef BattledexList<WaterPet>::Pets Battledex_water;
	for (const Battledex_water* pn = &Water_Battledex.getNext(); pn !=
&Battledex_water::NIL; pn = &pn->getNext())
	{
		if (pn->getPet().getName() == pokemon)
		{
			return pn->getNext().getPet();
		}
	}
}

BattledexList<FirePet>::Pets Battledex::PreviousFirePet(string pokemon)
{
	typedef BattledexList<FirePet>::Pets Battledex_fire;
	for (const Battledex_fire* pn = &Fire_Battledex.getNext(); pn !=
&Battledex_fire::NIL; pn = &pn->getNext())
	{
		if (pn->getPet().getName() == pokemon)
		{
			return pn->getPrevious().getPet();
		}
	}
}
```

```
BattledexList<FirePet>::Pets Battledex::NextFirePet(string pokemon)
{
	typedef BattledexList<FirePet>::Pets Battledex_fire;
	for (const Battledex_fire* pn = &Fire_Battledex.getNext(); pn !=
&Battledex_fire::NIL; pn = &pn->getNext())
	{
		if (pn->getPet().getName() == pokemon)
		{
			return pn->getNext().getPet();
		}
	}
}
BattledexList<GrassPet>::Pets Battledex::PreviousGrassPet(string pokemon)
{
	typedef BattledexList<GrassPet>::Pets Battledex_grass;
	for (const Battledex_grass* pn = &Grass_Battledex.getNext(); pn !=
&Battledex_grass::NIL; pn = &pn->getNext())
	{
		if (pn->getPet().getName() == pokemon)
		{
			return pn->getPrevious().getPet();
		}
	}
}

BattledexList<GrassPet>::Pets Battledex::NextGrassPet(string pokemon)
{
	typedef BattledexList<GrassPet>::Pets Battledex_grass;
	for (const Battledex_grass* pn = &Grass_Battledex.getNext(); pn !=
&Battledex_grass::NIL; pn = &pn->getNext())
	{
		if (pn->getPet().getName() == pokemon)
		{
			return pn->getNext().getPet();
		}
	}
}

void Battledex::printWaterPet()
{
	typedef BattledexList<WaterPet>::Pets Battledex_water;
	for (const Battledex_water* pn = &Water_Battledex.getNext(); pn !=
&Battledex_water::NIL; pn = &pn->getNext())
	{
		cout << "ID: " << pn->getPet().getID() << "\tName: " <<
pn->getPet().getName() << endl;
	}
}

void Battledex::printFirePet()
{
	typedef BattledexList<FirePet>::Pets Battledex_fire;

	for (const Battledex_fire* pn = &Fire_Battledex.getNext(); pn !=
&Battledex_fire::NIL; pn = &pn->getNext())
	{
		cout << "ID: " << pn->getPet().getID() << "\tName: " <<
pn->getPet().getName() << endl;
	}
}
```

Chan Kwang Yung 101215067

```
void Battledex::printGrassPet()
{
	typedef BattledexList<GrassPet>::Pets Battledex_grass;

	for (const Battledex_grass* pn = &Grass_Battledex.getNext(); pn != 
&Battledex_grass::NIL; pn = &pn->getNext())
	{
		cout << "ID: " << pn->getPet().getID() << "\tName: " << 
pn->getPet().getName() << endl;
	}
}
```

Chan Kwang Yung 101215067

BattledexList.h

```
#pragma once
template<class T>
class BattledexList
{//this class is used as the doubly linked list for the different type of Pets
	//it involves template class as well
public:
	typedef BattledexList<T> Pets;
private:
	T fPet;
	Pets* fNext;
	Pets* fPrevious;

public:
	BattledexList()
	{
		fPet = T();
		fNext = &NIL;
		fPrevious = &NIL;
	}//default constructor
	static Pets NIL;

	BattledexList(T aPet);//overload constructor

	//appending and prepending the nodes into the linked list
	void prepend(Pets& aPets);
	void append(Pets& aPets);
	void remove();//removes nodes from the linked list

	T getPet() const//gets the data in the current node
	{
		return fPet;
	}
	Pets& getNext() const
	{
		return *fNext;
	}
	Pets& getPrevious() const
	{
		return *fPrevious;
	}
};

template<class T>
BattledexList<T> BattledexList<T>::NIL;

template<class T>
BattledexList<T>::BattledexList(T aPet)
{
	fPet = aPet;
	fNext = &NIL;
	fPrevious = &NIL;
}

template<class T>
void BattledexList<T>::prepend(Pets& aPets)
{
	aPets.fNext = this; // points this as the next node of aPets
	if (fPrevious != &NIL)
		//this makes it so the current node is the backwards pointer of aPet and
if pointer forwards, it will be aPet
```

```
	{
		aPets.fPrevious = fPrevious;
		fPrevious->fNext = &aPets;
	}
	fPrevious = &aPets;
}

template<class T>
void BattledexList<T>::append(Pets& aPets)
{
	aPets.fPrevious = this; // points this as the previous node of aPets
	if (fNext != &NIL)
		//this makes it so the current node is the forwards pointer of aPets
which will be behind aPet
	{
		aPets.fNext = fNext;
		fNext->fPrevious = &aPets;
	}
	fNext = &aPets;
}

template<class T>
void BattledexList<T>::remove()
{
	if (fNext == &NIL)
	{
		fPrevious->fNext = &NIL;
	}
	else if (fPrevious == &NIL)
	{
		fNext->fPrevious = &NIL;
	}
	else
	{
		fPrevious->fNext = fNext;
		fNext->fPrevious = fPrevious;
	}

	//delete this;
}
```

Chan Kwang Yung 101215067

FirePet.h

```
#pragma once
#include "Pets.h"
class FirePet :public Pets //this class inherits from the Pets Class
{
private:
	string type;
public:
	FirePet();
	FirePet(int ID, string pokename, int hp, int currenthp, Moves movea, Moves moveb, Moves movec, Moves moved);
	//function overriding to signify polymorphism
	void printType();
	~FirePet();//destructor

};
```

FirePet.cpp

```
#include "FirePet.h"
FirePet::FirePet() //default constructo do not possess any actions
{

}

FirePet::FirePet(int ID, string petname, int hp, int currenthp, Moves movea, Moves moveb, Moves movec, Moves moved) :Pets(ID, petname, hp, currenthp, movea, moveb, movec, moved) //inheriting the fields from its parent class
{
	type = "Fire";
}

void FirePet::printType()
{
	cout << " \tFire Type" << endl;//this prints the type of the pets (fire) which overrides the function in its parent class
}

FirePet::~FirePet()
{

}
```

Chan Kwang Yung 101215067

GrassPet.h

```
#pragma once
#include "Pets.h"
class GrassPet :public Pets//this class inherits from the Pokemon Class
{
private:
	string type;
public:
	GrassPet();
	GrassPet(int ID, string petname, int hp, int currenthp, Moves movea, Moves
moveb, Moves movec, Moves moved);
	//function overriding to signify polymorphism
	void printType();
	~GrassPet();//destructor

};
```

GrassPet.cpp

```
#include "GrassPet.h"
GrassPet::GrassPet() //default constructor do not possess any actions
{

}

GrassPet::GrassPet(int ID, string petname, int hp, int currenthp, Moves movea, Moves
moveb, Moves movec, Moves moved) :Pets(ID, petname, hp, currenthp, movea, moveb,
movec, moved)//inheriting the fields from its parent class
{
	type = "Grass";
}

void GrassPet::printType()
{
	cout << " \tGrass Type" << endl;//this prints the type of the pets (grass)
which overrides the function in its parent class
}

GrassPet::~GrassPet()
{

}
```

Chan Kwang Yung 101215067

Moves.h

```h
#pragma once
#include <iostream>
using namespace std;
class Moves //this class is used to create moves for the Pets to use
{
private:

public:
	string moveName;
	int damage;
	string moveType;
	Moves();
	Moves(string name, int dmg, string type);//overload constructor
	//getters
	string getName();
	string getType();
	int getdmg();
	~Moves();
};
```

Moves.cpp

```cpp
#include "Moves.h"
Moves::Moves()
{

}

Moves::Moves(string name, int dmg, string type)
{
	moveName = name;
	damage = dmg;
	moveType = type;
}

string Moves::getName()
{
	return moveName;
}

string Moves::getType()
{
	return moveType;
}

int Moves::getdmg()
{
	return damage;

}
Moves::~Moves() {}
```

Chan Kwang Yung 101215067

MovesIterator.h

```
#pragma once
#include "MovesTemplate.h"
#include <iostream>

using namespace std;
//this class uses a template class and is used as an iterator for the singly link list
template <class T>
class MovesIterator
{
private:
	MovesNode<T>* fNode; //this stores the data for each node in the singly link
list
public:
	typedef MovesIterator<T> Iterator; //defines the name to describe
MovesIterator<T> as Iterator for easier use

	MovesIterator(MovesNode<T>* aNode);//default constructor

	//an overload operator for pointers where it will return the address of the
data from the node
	const T& operator*() const;

	//overload operators used for iterating through the nodes through incrementing
	//this also contains both prefix and postfix methods
	Iterator& operator++();
	Iterator operator++(int);

	//overload comparison operators used to compare between different nodes to see
whether they are equal or not
	bool operator==(const Iterator& aOther) const;
	bool operator!=(const Iterator& aOther) const;

	//detects the end of the link
	Iterator end();

	T getMoves() const
	{
		return fNode->fMoves;
	}
};

//implementation is done inside the header file
template <class T>
MovesIterator<T>::MovesIterator(MovesNode<T>* aNode)
{
	fNode = aNode;
}

//all the implementation needs to have the template <class T> declaration in order to
work
template <class T>
const T& MovesIterator<T>::operator*() const
{
	return fNode->fMoves;
}

//this is the prefix incremental operator
template <class T>
MovesIterator<T>& MovesIterator<T>::operator++()
{
```

```
	fNode = fNode->fNext; //moves from one node to the next node and returns it
	return *this;
}

//this is the postfix incremental operator
template <class T>
MovesIterator<T> MovesIterator<T>::operator++(int)
{
	MovesIterator<T> temp = *this;
	fNode = fNode->fNext;
	return temp;
}

//comparison overload operators
template <class T>
bool MovesIterator<T>::operator==(const Iterator& aOther) const
{
	return (fNode == aOther.fNode);
}

template <class T>
bool MovesIterator<T>::operator!=(const Iterator& aOther) const
{
	return !(*this == aOther);
}

//detects the end of the singly link list
template <class T>
MovesIterator<T> MovesIterator<T>::end()
{
	MovesIterator<T> temp = *this;
	while (temp.fNode != NULL)
	{
		temp++;
	}//keeps increasing the node until a NULL is met
	return temp;
}
```

Chan Kwang Yung 101215067

MovesTemplate.h

```
#pragma once

template <class T>
class MovesNode
{//this class is used for creating the singly link list nodes
public:
	T fMoves;
	MovesNode* fNext;
	MovesNode(const T& aMoves, MovesNode* aNext = (MovesNode*)0)
		//this is the constructor used to create the nodes and also link the
next node to it
	{
		fMoves = aMoves;
		fNext = aNext;
	}
};
```

Chan Kwang Yung 101215067

Owner.h

```
#pragma once
#include <iostream>
#include <cstdlib>
#include "Pets.h"
#include "Moves.h"
#include "PetStack.h"
using namespace std;
//this class is used to create the trainer's Pets team which is stored in a queue
class Owner
{
private:
	string name;
	Pets* petTeam;
	//the fields below are used later on in the implementation to access the queue
	int capacity;
	int front;
	int rear;
	int count;

public:
	Owner();
	Owner(string n);
	void setName(string n);
	string getName();
	//queue implementations
	void enqueue(Pets Pets);
	void dequeue();
	Pets& peek();
	int getTeamSize();
	bool isEmpty();
	bool isFull();
	~Owner();
};
```

Chan Kwang Yung 101215067

Owner.cpp

```
#include "Owner.h"
Owner::Owner()
{
    petTeam = new Pets[3];
    capacity = 3;
    front = 0;
    rear = -1;
    count = 0;
}
Owner::Owner(string n)
{
    name = n;
    petTeam = new Pets[3];
    capacity = 3;
    front = 0;
    rear = -1;
    count = 0;
}

void Owner::setName(string n)
{
    name = n;
}

string Owner::getName()
{
    return name;
}

//adding elements into the queue
void Owner::enqueue(Pets pets)
{
    if (isFull())
    {
        cout << "Your pets are at full capacity!" << endl;
    }
    //calculates the last index inside of the queue and then places the element into
that index
    rear = (rear + 1) % capacity;
    petTeam[rear] = pets;
    count++;//increase count which will be used to getTeamSize

}

//gets rid elements inside of the queue one by one
void Owner::dequeue()
{
    if (isEmpty())
    {
        cout << "No pets in the queue!" << endl;
    }

    front = (front + 1) % capacity;
    count--;


}

//this looks at the first element of the queue
Pets& Owner::peek()
```

```
{
    if (isEmpty())
    {
        cout << "No pets in the queue!" << endl;
    }
    return petTeam[front];
}

//returns count which is every time a enqueue or dequeu happens
int Owner::getTeamSize()
{
    return count;
}

//if the team size equals to 0, return true
bool Owner::isEmpty()
{
    return (getTeamSize() == 0);
}

//if the team size is equal to capacity, return true
bool Owner::isFull()
{
    return (getTeamSize() == capacity);
}

Owner::~Owner() {}
```

Chan Kwang Yung 101215067

Pets.h

```
#pragma once
using namespace std;
#include <iostream>
#include "Moves.h"
//this is the parent class of the derived classes
class Pets
{
private:
	int petID;
	string name;
	int MaxHP;
	int CurrentHP;
	Moves move1;
	Moves move2;
	Moves move3;
	Moves move4;


public:
	//this class contains most of the methods that are going to be used by the
child classes such as the getters and setters
	Pets();
	Pets(int ID, string petname, int hp, int currenthp, Moves movea, Moves moveb,
Moves movec, Moves moved);
	string getName();
	int getID();
	void setHP(int hp);
	int getMaxHP();
	int getCurrentHP();
	Moves getMove(int moveNo);
	virtual void printType();
	~Pets();
};
```

Chan Kwang Yung 101215067

Pets.cpp

```
#include "Pets.h"
#include <iostream>
using namespace std;

Pets::Pets() {}

//overload constructor
Pets::Pets(int ID, string petname, int hp, int currenthp, Moves movea, Moves moveb,
Moves movec, Moves moved)
{
	petID = ID;
	name = petname;
	MaxHP = hp;
	CurrentHP = currenthp;
	move1 = movea;
	move2 = moveb;
	move3 = movec;
	move4 = moved;

}

//used in polymorphism
void Pets::printType()
{

}

string Pets::getName()
{
	return name;
}

int Pets::getID()
{
	return petID;
}

//the setHP function is used to change the hp of the Pets
void Pets::setHP(int hp)
{
	CurrentHP = CurrentHP - hp;
}

int Pets::getMaxHP()
{
	return MaxHP;
}

int Pets::getCurrentHP()
{
	return CurrentHP;
}

//this getter is used to return the moves of the Pets based on the move number
Moves Pets::getMove(int moveNo)
{
	Moves move;
	if (moveNo == 1) {
		move = move1;
	}
```

```
	else if (moveNo == 2) {
		move = move2;
	}
	else if (moveNo == 3) {
		move = move3;
	}
	else if (moveNo == 4) {
		move = move4;
	}
	return move;
}

Pets::~Pets() {}
```

Chan Kwang Yung 101215067

PetStack.h

```
#pragma once
#include <iostream>
using namespace std;
//this class is used to represent the stack concept which is used in the main for how
the enemy pokemon team will be ordered by
template <class T>
class PetStack
{
private:
	T* PetElements;
	int PetStackPointer;
	int PetStackSize;
public:
	PetStack(int size);
	~PetStack();

	//this checks if the stack is empty
	bool isEmpty() const;

	//this checks the size of the stack
	int size() const;

	//this pushes objects into the stack
	void push(T aItem);

	//this removes the 1st element on top of the stack which is the last element to
enter the stack
	void pop();

	//this reads the top of the stack
	T& top();

};

template <class T>
PetStack<T>::PetStack(int size)
{//initializes the stack
	if (size <= 0)//if the size from the parameter is smaller than 0, then throw a
error message
	{
		throw invalid_argument("Error: Invalid stack size. ");
	}
	else
	{
		PetElements = new T[size];
		PetStackPointer = 0;
		PetStackSize = size;
	}
}

//stack destructor
template <class T>
PetStack<T>::~PetStack()
{
	delete[] PetElements;
}

template <class T>
bool PetStack<T>::isEmpty() const
{
```

```
	return PetStackPointer == 0;//once the stack pointer hits is 0 which means the
size of the stack is 0
}

template <class T>
int PetStack<T>::size() const
{
	return PetStackPointer;//returns the current position the stack pointer is
pointing at the top of the stack
}

template <class T>
void PetStack<T>::push(T aItem)
{
	if (PetStackPointer < PetStackSize)//if the current size is smaller than the
size set when creating the stack
	{
		PetElements[PetStackPointer++] = aItem;//assigns the parsed in item into
available slot
	}
	else
	{
		throw overflow_error("Stack has reached maximum capacity.");
	}
}

template <class T>
void PetStack<T>::pop()
{
	if (!isEmpty())//if the stack is not empty
	{
		PetStackPointer--;//removes the stack pointer starting from the top
	}
	else
	{
		throw underflow_error("Stack is currently empty.");
	}
}

template <class T>
T& PetStack<T>::top() //returns the item in the current slot of the stack which is at
the top
{
	if (!isEmpty())
	{
		return PetElements[PetStackPointer - 1];
	}
	else
	{
		throw underflow_error("Stack is currently empty.");
	}
}
```

Chan Kwang Yung 101215067

PetTree.h

```
#pragma once
#include <stdexcept>
#include "PetVisitor.h"
using namespace std;

template<class T>
class PetTree {
private:
	const T* fKey;
	PetTree <T>* fLeft;
	PetTree <T>* fRight;

	PetTree() :fKey((T*)0)
	{
		fLeft = &NIL;
		fRight = &NIL;
	}
public:
	static PetTree<T> NIL; //sentinel
	PetTree(const T& aKey) : fKey(&aKey)
	{
		fLeft = &NIL;
		fRight = &NIL;
	}//(complete this)

	~PetTree()
	{
		if (fLeft != &NIL)
			delete fLeft;
		if (fRight != &NIL)
			delete fRight;
	}//(complete this)

	bool isEmpty() const
	{
		return this == &NIL;
	}//(complete this)

	const T& key() const
	{
		if (isEmpty())
			throw domain_error("Empty PetTree");
		return *fKey;
	}//(complete this)

	PetTree& left() const
	{
		if (isEmpty())
			throw domain_error("Empty PetTree");
		return *fLeft;
	}

	PetTree& right() const
	{
		if (isEmpty())
			throw std::domain_error("Empty PetTree");
		return *fRight;
	}

	void attachLeft(PetTree<T>* aPetTree)
```

```
	{
		if (isEmpty())
			throw domain_error("Empty PetTree");
		if (fLeft != &NIL)
			throw domain_error("Non-empty sub tree");
		fLeft = new PetTree<T>(*aPetTree);
	}//(complete this)

	void attachRight(PetTree<T>* aPetTree) {
		if (isEmpty())
			throw std::domain_error("Empty PetTree");
		if (fRight != &NIL)
			throw std::domain_error("Non-empty sub tree");
		fRight = new PetTree<T>(*aPetTree);//makes allocation on heap

	}
	PetTree* detachLeft()
	{
		if (isEmpty())
			throw std::domain_error("Empty PetTree");
		PetTree<T>& Result = *fLeft; //changed to pointer variable
		fLeft = &NIL;
		return &Result;
	}

	PetTree* detachRight() {
		if (isEmpty())
			throw std::domain_error("Empty PetTree");
		PetTree<T>& Result = *fRight; //changed to pointer variable
		fRight = &NIL;
		return &Result;
	}

	//calls the visit method based on the type of class parsed in and moves to the
next node and does a recurssion
	void accept(const PetVisitor<T>& aVisitor) const
	{
		if (!isEmpty())
		{

			aVisitor.visit(key());
			left().accept(aVisitor);
			right().accept(aVisitor);
		}
	}
};

template<class T>
PetTree<T> PetTree<T>::NIL;
```

Chan Kwang Yung 101215067

PetVisitor.h

```
#pragma once
#include <iostream>
#include <fstream>
using namespace std;
template <class T>
class PetVisitor {
public:
	virtual ~PetVisitor() {} //virtual default destructor

	virtual void visit(const T& aKey) const{}
};

template <class T>
class PreviewVisitor : public PetVisitor<T> {
public:
	//this visit is used to print out all the keys inside of the tree
	virtual void visit(const T& aKey) const
	{
		cout << aKey << " ";
	}
};

template <class T>
class DisplayVisitor : public PetVisitor<T> {
public:
	//this visit is used to write data into a text file
	virtual void visit(const T& aKey) const
	{

		ofstream myfile;
		myfile.open("BattlePetsData.txt", ios_base::app);
		myfile << aKey << "\n";
		myfile.close();
	}
};
```

Chan Kwang Yung 101215067

WaterPet.h

```
#pragma once
#include "Pets.h"
class WaterPet :public Pets//this class inherits from the Pokemon Class
{
private:
	string type;

public:
	WaterPet();
	WaterPet(int ID, string petname, int hp, int currenthp, Moves movea, Moves
moveb, Moves movec, Moves moved);
	//function overriding to signify polymorphism
	void printType();
	~WaterPet();//destructor

};
```

WaterPet.cpp

```
#include "WaterPet.h"
WaterPet::WaterPet() //default constructor do not possess any actions
{

}

WaterPet::WaterPet(int ID, string petname, int hp, int currenthp, Moves movea, Moves
moveb, Moves movec, Moves moved) :Pets(ID, petname, hp, currenthp, movea, moveb,
movec, moved)//inheriting the fields from its parent class
{
	type = "Water";
}

void WaterPet::printType()
{
	cout << " \tWater Type" << endl;//this prints the type of the pet (water) which
overrides the function in its parent class
}

WaterPet::~WaterPet()
{

}
```