BLG336E ANLYSIS OF ALGORITHM 2

HOMEWORK – 1

Melik Mehmet BIYIK

150160534

23.03.2020

In this homework we are expected to simulate a pokemon battle. The selected pokemons are “Pikachu” and “Baltoise”.

Pikachu’s and Baltoise’s hp, pp, and attack infos are given in the txt files. Each pokemon has 3 attack and 1 skip attack.

At the begginning of the source.cpp I started with including necessary libraries which are string and vector libraries.

```
#include<stdio.h>  //(Standard input-output header)
#include<string.h> //(String header)
#include <vector>
#include <string>
```

After that I write Classes inside the source.cpp.

Firstly I generate a class for the attack named “Attack”.

```
//Generating CLASS for Attack
class Attack
{
public:
    string Name;
    int PP;
    int Accuracy;
    int Damage;
    int FirstUsage;
};
```

This class has attributes like ;

- Name -> Name of the attack
- PP -> Pikachu Point
- Accuracy -> percentage of the attack hit
- Damage -> attacks damage
- FirstUsage -> The level of grafh when this attack can be used.

After that I generate a class for Pokemons named “Pokemon”.

```
19      //Generating Class for Pokemon
20     class Pokemon
21     {
22     public:
23         string Name;
24         int HP;
25         int PP;
26         Attack Attacks[4];
27     };
```

This class has attributes like;

- Name -> name of the pokemon
- HP -> Health Point of the pokemon
- PP -> Pokemon Point of the pokemon
- Attacs[4] -> A list of attack(3 ability and skip for each pokemon)

After the generation of Pokemon Class, I write another class for node representation.

```
//Generating Class for Graph's Node
class Node
{
public:
    int ID;
    Pokemon Pikachu;
    Pokemon Blastoise;
    char Turn;
    double Porbablity;
    int Level;
    bool IsLeaf;
```

This class has attributes like;

- ID -> for controlling and monitoring the nodes
- Pikachu -> Fighting pokemon(inherited from Pokemon Class)
- Blastoise -> Other pokemon
- Turn -> Determines the attack turn (‘P’ and ‘B’)
- Probablity -> the value of probablity fort his node
- Level -> the level of the graph
- IsLeaf -> Flag for the isLeaf node

The Necessary classes are genareted.

In main method I started with generating first pokemon called Pikachu.

```
int main() {

    //Generating the Pokemon - PIKACHU
    Pokemon pikachu;
```

It is the object of the Pokemon class.

After generating the object I filled its attributes.

```
    pikachu.Name = "Pikachu"; pikachu.HP = 150; pikachu.PP = 100;
    //Adding attacks to Pikachu
    pikachu.Attacks[0].Name = "Thundershock"; pikachu.Attacks[0].PP = -10; pikachu.Attacks[0].Accuracy = 100; pikachu.Attacks[0].Damage = 40; pikachu.Attacks[0].FirstUsage = 0;
    pikachu.Attacks[1].Name = "Skull Bash"; pikachu.Attacks[1].PP = -15; pikachu.Attacks[1].Accuracy = 70; pikachu.Attacks[1].Damage = 50; pikachu.Attacks[1].FirstUsage = 0;
    pikachu.Attacks[2].Name = "Slam"; pikachu.Attacks[2].PP = -20; pikachu.Attacks[2].Accuracy = 80; pikachu.Attacks[2].Damage = 60; pikachu.Attacks[2].FirstUsage = 0;
    pikachu.Attacks[3].Name = "Skip"; pikachu.Attacks[3].PP = 100; pikachu.Attacks[3].Accuracy = 100; pikachu.Attacks[3].Damage = 0; pikachu.Attacks[3].FirstUsage = 3;
```

In here all attributes and attackList are filled for “Pikachu”.

And same steps for the other pokemon ; “Blastoise”.

```
    //Generating the Pokemon - BLASTOISE
    Pokemon blastoise;

    blastoise.Name = "Blastoise"; blastoise.HP = 150; blastoise.PP = 100;
    //Adding attacks to Blastoise
    blastoise.Attacks[0].Name = "Tackle"; blastoise.Attacks[0].PP = -10; blastoise.Attacks[0].Accuracy = 100; blastoise.Attacks[0].Damage = 30; blastoise.Attacks[0].FirstUsage = 0;
    blastoise.Attacks[1].Name = "Water Gun"; blastoise.Attacks[1].PP = -20; blastoise.Attacks[1].Accuracy = 100; blastoise.Attacks[1].Damage = 40; blastoise.Attacks[1].FirstUsage = 0;
    blastoise.Attacks[2].Name = "Bite"; blastoise.Attacks[2].PP = -25; blastoise.Attacks[2].Accuracy = 100; blastoise.Attacks[2].Damage = 60; blastoise.Attacks[2].FirstUsage = 0;
    blastoise.Attacks[3].Name = "Skip"; blastoise.Attacks[3].PP = 100; blastoise.Attacks[3].Accuracy = 100; blastoise.Attacks[3].Damage = 0; blastoise.Attacks[3].FirstUsage = 3;
```

After generating the pokemons I inherit an object from Node class. This is my first Node.

```
//Generating the first node of the graph
Node firstNode;
```

And filled its attributes; It represent the beginning of the fight.

```
//filling the attributes of firstNode object
firstNode.ID = 0;
firstNode.Pikachu = pikachu;
firstNode.Blastoise = blastoise;
firstNode.Turn = 'P';
firstNode.Level = 0;
firstNode.IsLeaf = 0;
firstNode.Porbablity = 1;
```

After that I generated a vector for holding the node Lists and I pushed the first node in it.

```
    vector<Node> nodeList;
    nodeList.push_back(firstNode);
```

After that I called the function for nextLevel and I passed the firstNode and nodeList as parameters.

In here I writed a method named “NextLevel” for simulating attack steps and possible Node cases.

```
// ------------------------------------ METHODS ------------------
vector<Node> NextLevel(Node parentNode, vector<Node> nodeList);
```

This method has 2 parameters: parentNode and the nodeList.

- parentNode holds the object of the previous node.
- nodeList is for adding the generated nodes in a list.

This method starts with control if the pikachu’s turn.

```
47      if (parentNode.Turn == 'P' && !parentNode.IsLeaf) { // Pikachu's Turn
48          for (int i = 0; i < 4; i++) {//4 loop for every attack possibilty
49              Node node; //child node
50              int idCounter = parentNode.ID + 1;
51              node.ID = idCounter;
52              node.Pikachu = parentNode.Pikachu;
53              node.Blastoise = parentNode.Blastoise;
54              node.Turn = parentNode.Turn;
55              node.Level = parentNode.Level;
56              node.IsLeaf = parentNode.IsLeaf;
57              node.Porbablity = parentNode.Porbablity;
```

In here I generated a child node and filled its attributes from its parent. I used for loop for representing the all atacks cases. In every loop it generates a node for its possiblity.

```
59              //calculate the current statistics
60              if (i == 0) {  //the Thunder Shock attack step
61                  node.Pikachu.PP = node.Pikachu.PP + node.Pikachu.Attacks[0].PP;
62                  if (node.Pikachu.PP >= 0) {  // if current PP is not less than 0(that means attack can be used)
63                      node.Blastoise.HP = node.Blastoise.HP - node.Pikachu.Attacks[0].Damage;
64                      node.Turn = 'B';
65                      node.Level = node.Level + 1;
66                      if (node.Level < 3) { node.Porbablity = node.Porbablity / double(3); }
67                      else { node.Porbablity = node.Porbablity / double(4); }//added the possibilty of skip attack
68                      if (node.Blastoise.HP <= 0) { node.IsLeaf = 1; } //Control of the K.O
69                      else { node.IsLeaf = 0; }
70
71                      nodeList.push_back(node); //pushing the new generated node to the nodeList
72                      NextLevel(node, nodeList); //recursively generate new nodes untill K.O
73                  }
74                  else { idCounter--; }
```

After that I write case for the first attack of pikachu. It has 100 accuracy so I did not need to generate mulitple nodes for this case.

Firstly I calculated the PP of pikachu.

I checked the pikachu’s PP is not 0.

I execute the attack to blastoise and calculation of Blastiose HP.

I switched the turn to blastoise.

I increment the level of the graph.

I calculated the probablity of the node.

I checked the Blastoise if knocked out.

And I pushed the current node to the NodeList.

And I called the function recursively.

After the case of first attack I started to write the second attack case of pikachu.

```
else if (i == 1) {//the Skull Bash attack step

    Node nodeMissSkull; //miss attack case node generated
    idCounter++;
    nodeMissSkull.ID = idCounter;
    nodeMissSkull.Pikachu = parentNode.Pikachu;
    nodeMissSkull.Blastoise = parentNode.Blastoise;
    nodeMissSkull.Turn = parentNode.Turn;
    nodeMissSkull.Level = parentNode.Level;
    nodeMissSkull.IsLeaf = parentNode.IsLeaf;
```

In here it has possiblity to miss (not 100 accuracy) so we need to consider it. So I generated 2 nodes fort hat. 1 node for hit attack and other node for nodeMissSkull attack.

```
            node.Pikachu.PP = node.Pikachu.PP + node.Pikachu.Attacks[1].PP;//pp calculated
            nodeMissSkull.Pikachu.PP = nodeMissSkull.Pikachu.PP + nodeMissSkull.Pikachu.Attacks[1].PP;
```

I calculated the PP of the pikachu for both nodes.

```
if (node.Pikachu.PP >= 0 && nodeMissSkull.Pikachu.PP >= 0) {  // if current PPs is not less than 0(that means attack can be used)
    node.Blastoise.HP = node.Blastoise.HP - node.Pikachu.Attacks[1].Damage; //no damage calc. for miss
    node.Turn = 'B';
    nodeMissSkull.Turn = 'B';
    node.Level = node.Level + 1;
    nodeMissSkull.Level = nodeMissSkull.Level + 1;
```

I checked the PP for the attack can happen or nott

And I calculated the new HP op Blastoise.

No need to calculate it for miss Node.

Switch the turn to ‘B’ for both.

Increment the levels.

```
if (node.Level < 3) { //no permission to skip (Same level for nodeMissSkull)
    node.Porbablity = (node.Porbablity * (node.Pikachu.Attacks[1].Accuracy / 100)) / double(3);
    nodeMissSkull.Porbablity = (nodeMissSkull.Porbablity * (double(1) - (nodeMissSkull.Pikachu.Attacks[1].Accuracy / 100))) / double(3); ///// Work On numbe
}
else {//added the possibilty of skip attack
    node.Porbablity = (node.Porbablity * (node.Pikachu.Attacks[1].Accuracy / 100)) / double(4);
    nodeMissSkull.Porbablity = (nodeMissSkull.Porbablity * (double(1) - (nodeMissSkull.Pikachu.Attacks[1].Accuracy / 100))) / double(4);
}
```

I calculated for both nodes for first 3 level and other levels. Controled it with ‘if’ ,’else’ controls.

After that I checked if Blastoise is dead or not. No need to control it for miss attack.

```
if (node.Blastoise.HP <= 0) { node.IsLeaf = 1; }
else { node.IsLeaf = 0; }
nodeMissSkull.IsLeaf = 0; // not possible to kill the Blastoise with missed attack
```

After that I pushed the nodes into the List.

```
nodeList.push_back(node);//Pushing the nodes to the nodeList
nodeList.push_back(nodeMissSkull);//multiple nodes added for miss
NextLevel(node, nodeList);//Recursively keep generating untill K.O
NextLevel(nodeMissSkull, nodeList);//Recursively keep generating untill K.O
```

I called the method recursively for next attacks.

I used same steps for the thirth attack with second attack of the pikachu.

```
else if (i == 2) {//The Slam Attack step
    Node nodeMissSlam; //miss attack case node generated
```

After that I started to skip attack case.

```
    else { // i == 3 so skip attack step
        if (node.Level >= 2 && node.Pikachu.PP < 100) {//check if the first usage passed
            node.Pikachu.PP = node.Pikachu.PP + node.Pikachu.Attacks[3].PP;// PP calculated
            node.Turn = 'B';//No need to control PP and calc blastoise.HP and no miss possibilty
            node.Level = node.Level + 1;
            node.Porbablity = (node.Porbablity * (node.Pikachu.Attacks[3].Accuracy / 100)) / double(4);//probablity of skip
            nodeList.push_back(node);//Pushing the node to the nodeList
            NextLevel(node, nodeList);//No way to K.O and IsLeaf
        }
    }
```

In here no way to k.o blastoise. In the end of this case pikachu's PP is added with 100.

After that the turn passed to Blastoise.

```
else if (parentNode.Turn == 'B' && !parentNode.IsLeaf) { //Blastoise's Turn
```

I also used for loops for each attack.

```
for (int i = 0; i < 4; i++) { //4 case for each attack
    Node node;//child node
    int idCounter = parentNode.ID + 1;
    node.ID = idCounter;
    idCounter++;
    node.Pikachu = parentNode.Pikachu;
    node.Blastoise = parentNode.Blastoise;
    node.Turn = parentNode.Turn;
    node.Level = parentNode.Level;
    node.IsLeaf = parentNode.IsLeaf;
    node.Porbablity = parentNode.Porbablity;
```

And I generated the node for this child step.

In here I did same things for Blastoise.

```
//calculate the current statistics
if (i == 0) {  //the Tackle attack step
    node.Blastoise.PP = node.Blastoise.PP + node.Blastoise.Attacks[0].PP;
    if (node.Blastoise.PP >= 0) {  // if current PP is not less than 0(that means attack can be used)
        node.Pikachu.HP = node.Pikachu.HP - node.Blastoise.Attacks[0].Damage;
        node.Turn = 'P';
        node.Level = node.Level + 1;
        if (node.Level < 3) { node.Porbablity = (node.Porbablity) / double(3); }//calc possibilty at first 3 step
        else { node.Porbablity = parentNode.Porbablity / double(4); }//added the possibilty of skip attack
        if (node.Pikachu.HP <= 0) { node.IsLeaf = 1; } //Control of the K.O
        else { node.IsLeaf = 0; }

        nodeList.push_back(node); //pushing the new generated node to the nodeList
        NextLevel(node, nodeList); //recursively generate new nodes untill K.O
    }
    else { idCounter--; }
}
```

I did not calculate the miss attack case because blastoise all attacks has 100 accuracy.

I recersively called the function for child nodes.

The *source.cpp* file can summarized liked this. I also annote for important lines in the code.