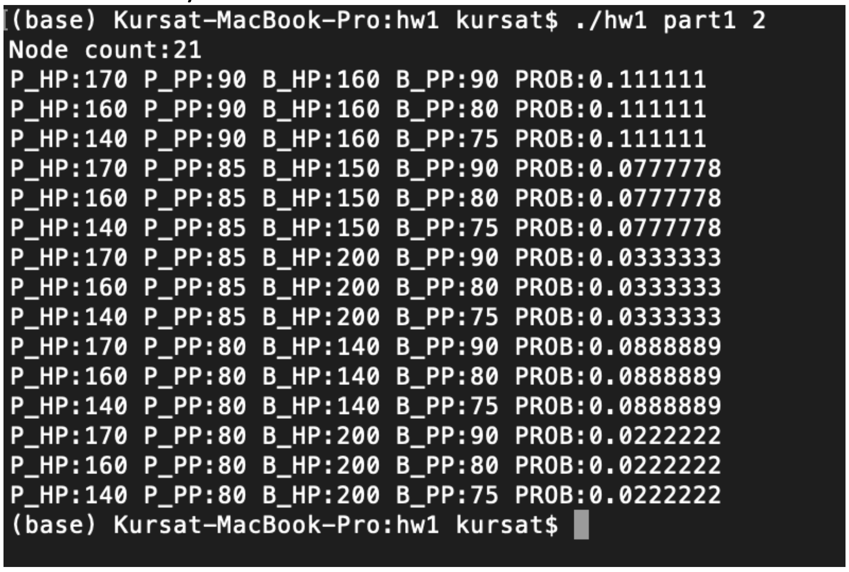
BLG336E Homework-1

O.Kürşat Karayılan
150140011

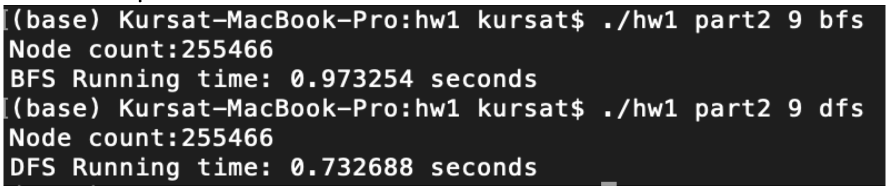
You can compile the code using command **`g++ -o hw1 hw1.cpp`**. Then you can run it with **`./hw1 part1 2`**

It should create following screen. Note: You need to have pikachu.txt and blastoise.txt in the same directory.



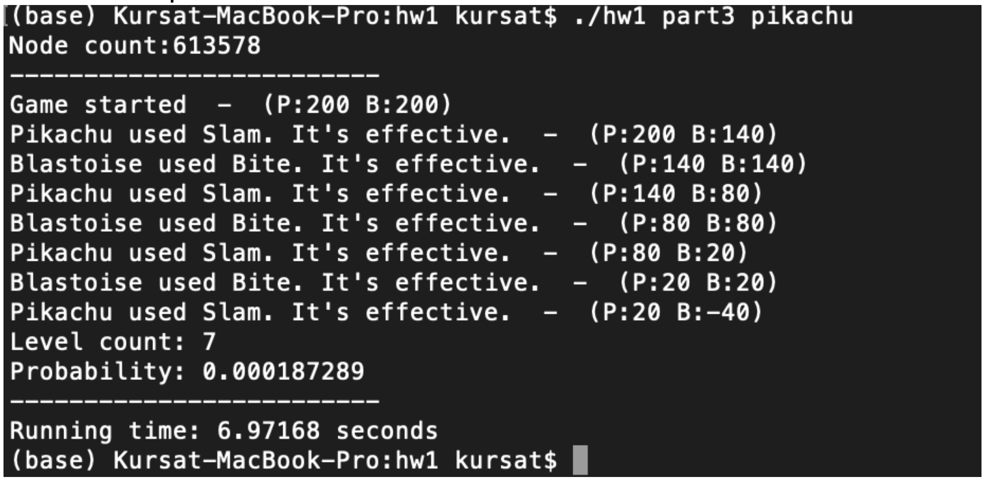
```
(base) Kursat-MacBook-Pro:hw1 kursat$ ./hw1 part1 2
Node count:21
P_HP:170 P_PP:90 B_HP:160 B_PP:90 PROB:0.111111
P_HP:160 P_PP:90 B_HP:160 B_PP:80 PROB:0.111111
P_HP:140 P_PP:90 B_HP:160 B_PP:75 PROB:0.111111
P_HP:170 P_PP:85 B_HP:150 B_PP:90 PROB:0.0777778
P_HP:160 P_PP:85 B_HP:150 B_PP:80 PROB:0.0777778
P_HP:140 P_PP:85 B_HP:150 B_PP:75 PROB:0.0777778
P_HP:170 P_PP:85 B_HP:200 B_PP:90 PROB:0.0333333
P_HP:160 P_PP:85 B_HP:200 B_PP:80 PROB:0.0333333
P_HP:140 P_PP:85 B_HP:200 B_PP:75 PROB:0.0333333
P_HP:170 P_PP:80 B_HP:140 B_PP:90 PROB:0.0888889
P_HP:160 P_PP:80 B_HP:140 B_PP:80 PROB:0.0888889
P_HP:140 P_PP:80 B_HP:140 B_PP:75 PROB:0.0888889
P_HP:170 P_PP:80 B_HP:200 B_PP:90 PROB:0.0222222
P_HP:160 P_PP:80 B_HP:200 B_PP:80 PROB:0.0222222
P_HP:140 P_PP:80 B_HP:200 B_PP:75 PROB:0.0222222
(base) Kursat-MacBook-Pro:hw1 kursat$
```

Part2 computes run times of bfs and dfs search.



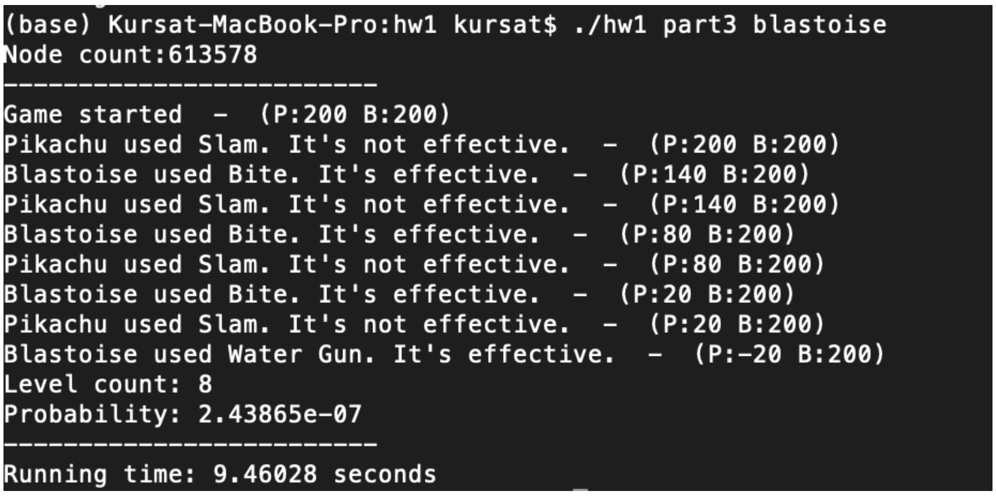
```
(base) Kursat-MacBook-Pro:hw1 kursat$ ./hw1 part2 9 bfs
Node count:255466
BFS Running time: 0.973254 seconds
(base) Kursat-MacBook-Pro:hw1 kursat$ ./hw1 part2 9 dfs
Node count:255466
DFS Running time: 0.732688 seconds
```

Part3 find easiest path with lowest level.
For the case pikachu wins.



```
(base) Kursat-MacBook-Pro:hw1 kursat$ ./hw1 part3 pikachu
Node count:613578
-------------------------
Game started  -  (P:200 B:200)
Pikachu used Slam. It's effective.  -  (P:200 B:140)
Blastoise used Bite. It's effective.  -  (P:140 B:140)
Pikachu used Slam. It's effective.  -  (P:140 B:80)
Blastoise used Bite. It's effective.  -  (P:80 B:80)
Pikachu used Slam. It's effective.  -  (P:80 B:20)
Blastoise used Bite. It's effective.  -  (P:20 B:20)
Pikachu used Slam. It's effective.  -  (P:20 B:-40)
Level count: 7
Probability: 0.000187289
-------------------------
Running time: 6.97168 seconds
(base) Kursat-MacBook-Pro:hw1 kursat$
```

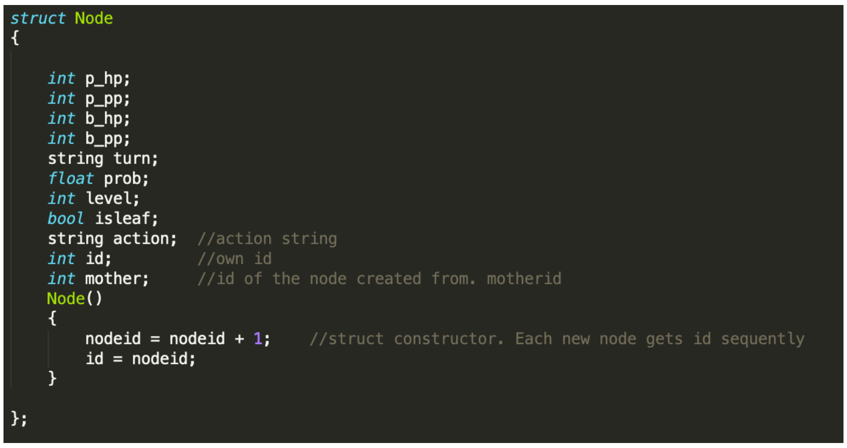
For the case Blastoise wins.



```
(base) Kursat-MacBook-Pro:hw1 kursat$ ./hw1 part3 blastoise
Node count:613578
-------------------------
Game started  -  (P:200 B:200)
Pikachu used Slam. It's not effective.  -  (P:200 B:200)
Blastoise used Bite. It's effective.  -  (P:140 B:200)
Pikachu used Slam. It's not effective.  -  (P:140 B:200)
Blastoise used Bite. It's effective.  -  (P:80 B:200)
Pikachu used Slam. It's not effective.  -  (P:80 B:200)
Blastoise used Bite. It's effective.  -  (P:20 B:200)
Pikachu used Slam. It's not effective.  -  (P:20 B:200)
Blastoise used Water Gun. It's effective.  -  (P:-20 B:200)
Level count: 8
Probability: 2.43865e-07
-------------------------
Running time: 9.46028 seconds
```

Graph Implementation

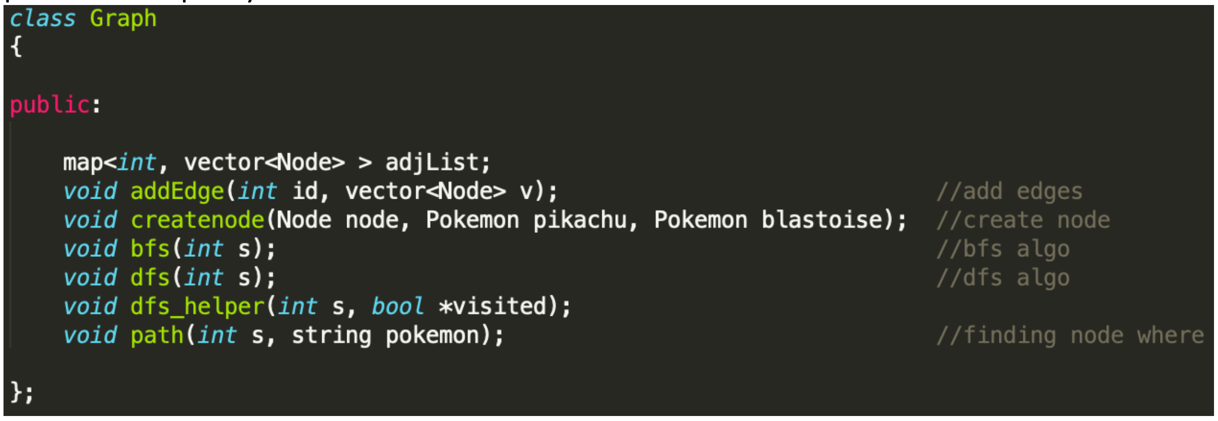
There is a struct called Node.



```
struct Node
{

    int p_hp;
    int p_pp;
    int b_hp;
    int b_pp;
    string turn;
    float prob;
    int level;
    bool isleaf;
    string action;  //action string
    int id;         //own id
    int mother;     //id of the node created from. motherid
    Node()
    {
        nodeid = nodeid + 1;    //struct constructor. Each new node gets id sequently
        id = nodeid;
    }

};
```

Each node gets a unique id starting from 0. Nodeid is a global variable. Nodeid keeps total node number. Also, each node has mother variable. Mother variable keeps the id of which node created this particular node. For example, level1 nodes have mother=0 since head node creates them.

Also, there is graph class. It’s not a good practice but I made all of methods and variables public for simplicity.

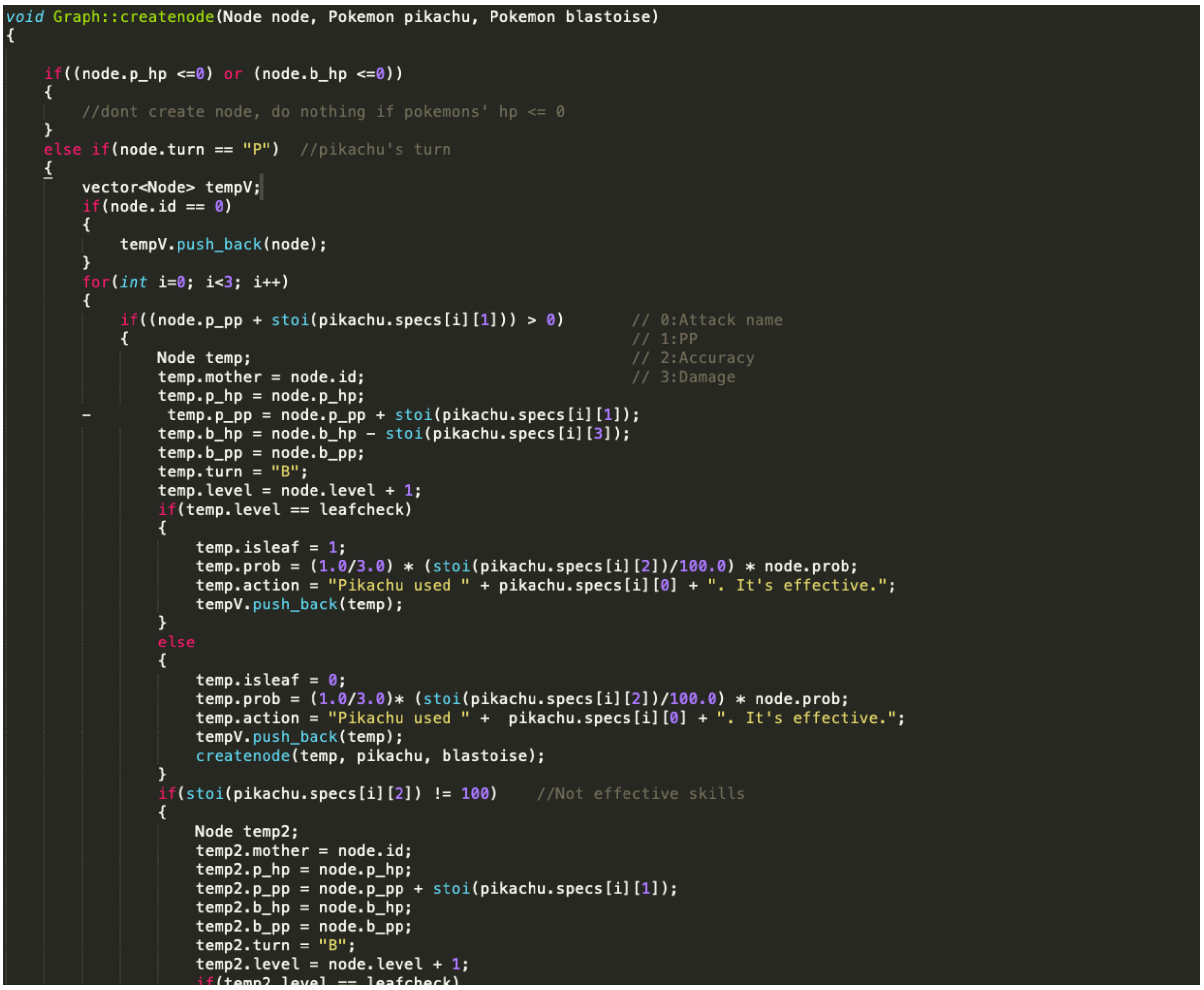


```
class Graph
{

public:

    map<int, vector<Node> > adjList;
    void addEdge(int id, vector<Node> v);                         //add edges
    void createnode(Node node, Pokemon pikachu, Pokemon blastoise); //create node
    void bfs(int s);                                              //bfs algo
    void dfs(int s);                                              //dfs algo
    void dfs_helper(int s, bool *visited);
    void path(int s, string pokemon);                             //finding node where

};
```

There is a map named adjList. It keeps nodes with their ids as keys. For example, adjList[0] keeps neighbors of head node. It's vector contains level1 and head nodes. I put head node into this vector as well.

- addEdge adds nodes into map. It takes id and vector.
- createnode function creates nodes.
- Bfs and dfs functions compute breath-first and depth-first searches.
- Dfs_helper function is part of the dfs method.
- Path function is for part3. It finds nodes with desired pokemon's hp=0 info using breath-first-search. Then, I choose lowest level one between them. Lastly printing action sequence.

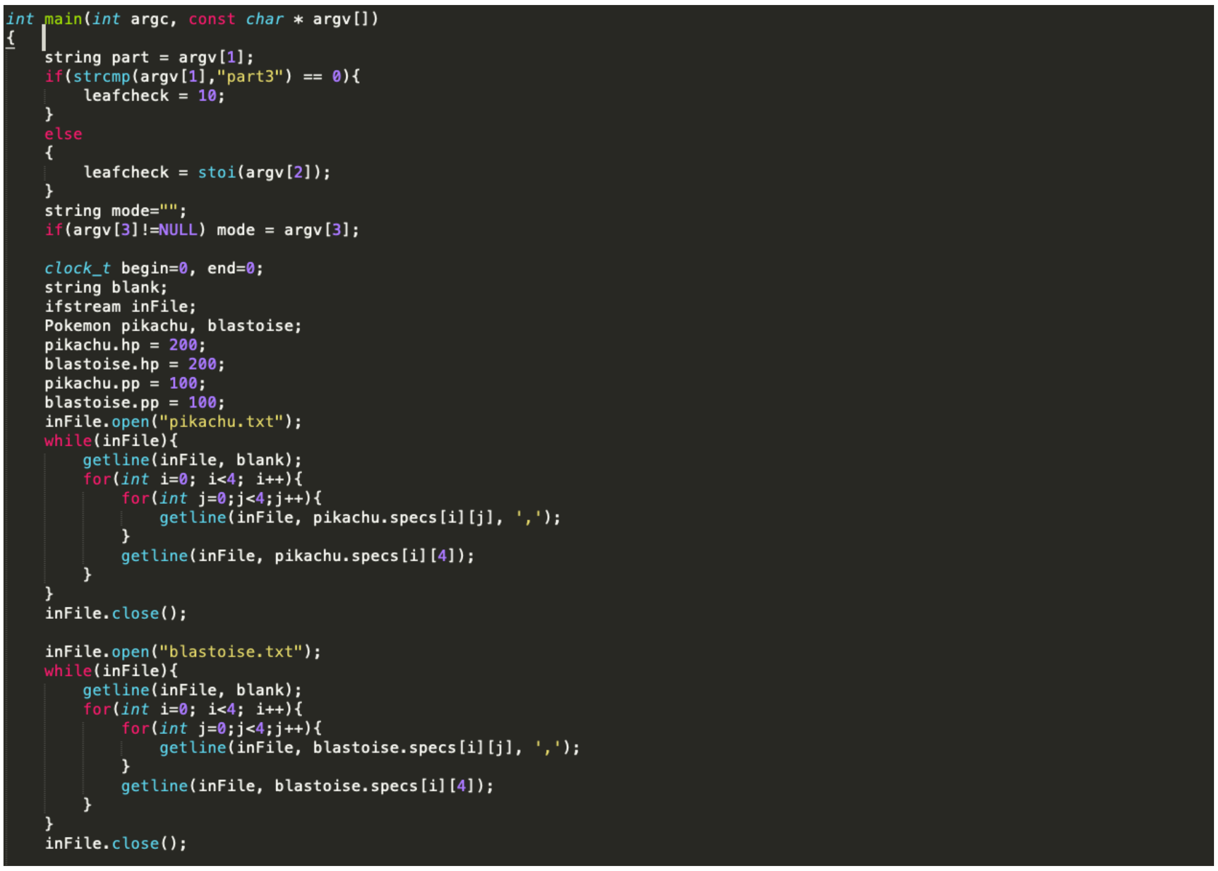
Createnode



```
void Graph::createnode(Node node, Pokemon pikachu, Pokemon blastoise)
{

    if((node.p_hp <=0) or (node.b_hp <=0))
    {
        //dont create node, do nothing if pokemons' hp <= 0
    }
    else if(node.turn == "P")  //pikachu's turn
    {
        vector<Node> tempV;
        if(node.id == 0)
        {
            tempV.push_back(node);
        }
        for(int i=0; i<3; i++)
        {
            if((node.p_pp + stoi(pikachu.specs[i][1])) > 0)       // 0:Attack name
            {                                                     // 1:PP
                Node temp;                                        // 2:Accuracy
                temp.mother = node.id;                            // 3:Damage
                temp.p_hp = node.p_hp;
                 temp.p_pp = node.p_pp + stoi(pikachu.specs[i][1]);
                temp.b_hp = node.b_hp - stoi(pikachu.specs[i][3]);
                temp.b_pp = node.b_pp;
                temp.turn = "B";
                temp.level = node.level + 1;
                if(temp.level == leafcheck)
                {
                    temp.isleaf = 1;
                    temp.prob = (1.0/3.0) * (stoi(pikachu.specs[i][2])/100.0) * node.prob;
                    temp.action = "Pikachu used " + pikachu.specs[i][0] + ". It's effective.";
                    tempV.push_back(temp);
                }
                else
                {
                    temp.isleaf = 0;
                    temp.prob = (1.0/3.0)* (stoi(pikachu.specs[i][2])/100.0) * node.prob;
                    temp.action = "Pikachu used " +  pikachu.specs[i][0] + ". It's effective.";
                    tempV.push_back(temp);
                    createnode(temp, pikachu, blastoise);
                }
                if(stoi(pikachu.specs[i][2]) != 100)    //Not effective skills
                {
                    Node temp2;
                    temp2.mother = node.id;
                    temp2.p_hp = node.p_hp;
                    temp2.p_pp = node.p_pp + stoi(pikachu.specs[i][1]);
                    temp2.b_hp = node.b_hp;
                    temp2.b_pp = node.b_pp;
                    temp2.turn = "B";
                    temp2.level = node.level + 1;
                    if(temp2.level == leafcheck)
```

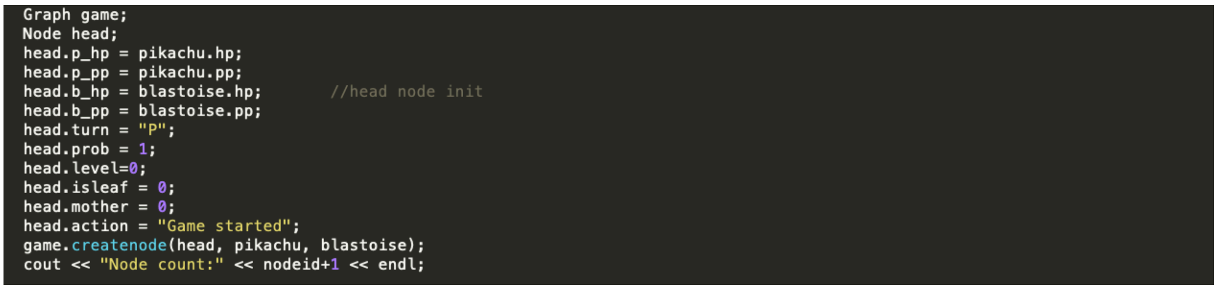
First if is for the case that if one of the pokemon's hp is below zero It does not create nodes from recursive calls. There is a for loop runs 3 times. It is 3 since there are three skills. I did not include skip skill at first. Then I could not implement later due to lack of time. I collect all skill nodes into tempV vector. Then at the end I call addEdge function to add into map. If particular node is not a leaf node, then it calls createnode function again with its information. If it is leafnode I check it with leafcheck variable. Leafcheck is a global variable and stores max level input.

Main



```cpp
int main(int argc, const char * argv[])
{
    string part = argv[1];
    if(strcmp(argv[1],"part3") == 0){
        leafcheck = 10;
    }
    else
    {
        leafcheck = stoi(argv[2]);
    }
    string mode="";
    if(argv[3]!=NULL) mode = argv[3];

    clock_t begin=0, end=0;
    string blank;
    ifstream inFile;
    Pokemon pikachu, blastoise;
    pikachu.hp = 200;
    blastoise.hp = 200;
    pikachu.pp = 100;
    blastoise.pp = 100;
    inFile.open("pikachu.txt");
    while(inFile){
        getline(inFile, blank);
        for(int i=0; i<4; i++){
            for(int j=0;j<4;j++){
                getline(inFile, pikachu.specs[i][j], ',');
            }
            getline(inFile, pikachu.specs[i][4]);
        }
    }
    inFile.close();

    inFile.open("blastoise.txt");
    while(inFile){
        getline(inFile, blank);
        for(int i=0; i<4; i++){
            for(int j=0;j<4;j++){
                getline(inFile, blastoise.specs[i][j], ',');
            }
            getline(inFile, blastoise.specs[i][4]);
        }
    }
    inFile.close();
```

For part3 I made level 10. Graph created for 10 levels. I also make pikachu and blastoise hp's 200 for faster runs.



```cpp
 Graph game;
 Node head;
 head.p_hp = pikachu.hp;
 head.p_pp = pikachu.pp;
 head.b_hp = blastoise.hp;        //head node init
 head.b_pp = blastoise.pp;
 head.turn = "P";
 head.prob = 1;
 head.level=0;
 head.isleaf = 0;
 head.mother = 0;
 head.action = "Game started";
 game.createnode(head, pikachu, blastoise);
 cout << "Node count:" << nodeid+1 << endl;
```

Then I create Graph object called game. I create head node and it's variables manually.