Tanay Cansin Dogan
150150119

Algorithm Analysis 2: Assignment1 – Spring 20 Aka. *"COVID-Term"*

Introduction:

In this assignment we have simulated a pokemon battle between Pikachu and Blastoise. During the simulation we have created a fight graph and then searched on this graph using breadth first search and depth first search algorithms. Besides that, during the implementation of the different parts of the assignment we have faced different problems such as allocation problems etc. and overcome those problems by developing our own algorithms/solutions.

The code is written in c++11 should be compiled with:

```
g++  -std=c++11 main.cpp –o project1
```

Classes:

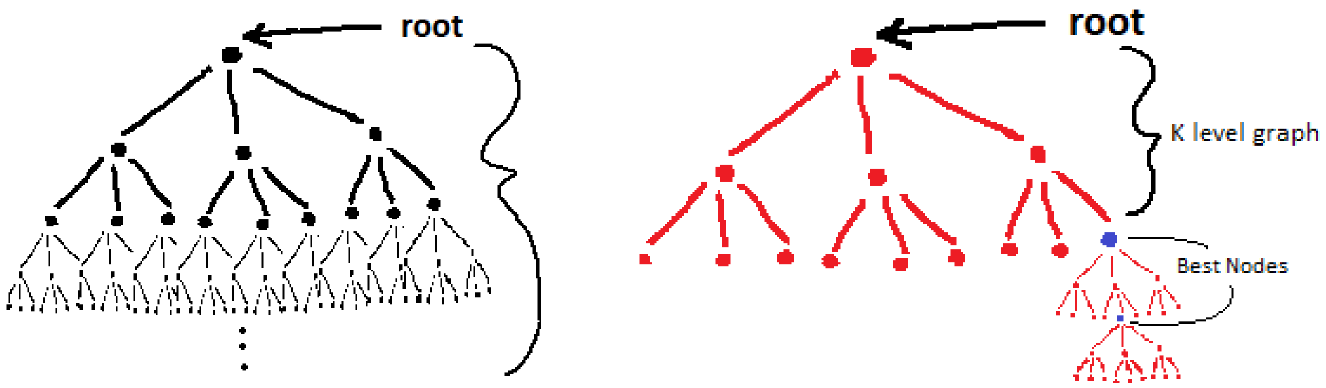
1. **Ability:** Represents different abilities of different pokemons, in our model of the program, instances of this class stored by **Pokemon** class's instances by a vector.
   - **Ability::Show():** is used for promt Attributes of the ability, this method is called in **Pokemon::Pokedex()** method.

2. **Pokemon:** Represents Pokemons (Pikachu and Blastoise in this case), in our model of the program, instances of this class passing through the **Node::Node()** by reference so they can help to create other nodes which represents the state that casting their different Abilities.
   - **Pokemon::Pokedex():** is used for promt Attributes of a Pokemon and its abilities.

3. **Node:** Represents each node of the Graph, in our model of the Program. The entire graph with K max-level is created with the creation of the root Node since **Node::Node()** is doing a recursive call. The successor Nodes are stored in a Node pointer vector since storing the instance would inflate memory usage.
   - **Node::Node():** Recursively create the whole graph whenever the root Node is created. Probability calculation is performed by the formula:

   Probability = (Parent's Probability)*(1/number of usable skills)*(Accuracy of used skill/100).
   - **Node::PromtMove():** Prompts the move that creates our current Node. Since root node doesn't have that kind of move, this method shouldn't use on the root of the Graph.
   - **Node::ShowNode() and Node::ShowGraph():** Helper methods that created in order to help debugging and the visualization of the Graph.
   - **Node::LastLayerInfo():** The method which traverse the whole graph recursively and outputs last's layers information as requested in the part2.
   - **Node::BFS():** the method where Breadth First Search algorithm is implemented iteratively. During the implementation queue data type is used since the shallow levels should have served first (FIFO) rather than the deeper levels.
   - **Node::DFS():** the method where Depth First Search algorithm is implemented iteratively. During the implementation stack data type is used since the shallow levels should have wait until deeper levels are served (LIFO).

4. **Match:** Represents the Pokemon Battle in our model of the Program. It consists of 2 Pokemon and then **Match::KnockOut()** method.
    - **Match::KnockOut():** Returns the Node that we searched for the opponent pokemon's HP reached 0. Since the Pikachu can win at first in 15th level –with blastoise Hp: 361, and Pikachu Hp: 273 values- and the creation of the 15 level graph resulted as memory allocation problems, I have solved that problem with developing a new algorithm. The algorithm searches **best node** –where opponent's HP is least and our pokemon's PP is greatest- in a **K** level graph, where **K** is between 3 and 10 since 10 is the maximum level that my computer can handle. The low **K** values are faster. And higher **K** values are more robust in case of using different set of abilities where ability combos matter. Then create a new **K** level graph from that previous iteration's search result and iterate till Opponent's HP reached 0. **Advantage:** In this



method we don't create the successors of nodes other than the **best node**