BLG 223E - Homework 2 Report

Melike Beşparmak - 150220061

May 2024

1 Introduction

The homework mostly focused on the Breadth-First Search (BFS) algorithm and required a graph implementation. The first part was about imitating every possible match between two pokemons, Pikachu and Blastoise. The second part aimed to find the shortest path to victory for both pokemons.

2 Methods

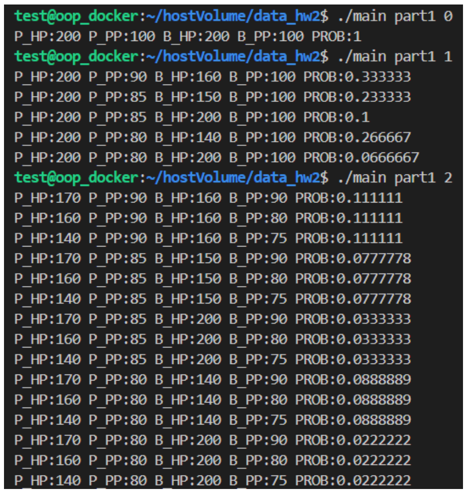
2.1 Part 1

To create the graph, I used a BFS algorithm using the provided Doubly Linked List header file and implemented a queue by using mainly *addBack* and *remove-Front* functions. I used two functions for this part which are *creategraph* and *part1*. The first one creates the graph of every possible attack until it reaches the maximum level provided as a parameter. As stated previously, the method used is a BFS algorithm and with this method, once a node is created it is added to the queue and popped after. For every iteration, the new nodes are added as children of the popped node. The new nodes are created according to possible attacks and effectiveness. Furthermore, the probability of a node is dependent on the parent node, and the accuracy. The necessary calculations are also computed during creation. Another critical point is that, during the creation of a new node, new pokemons are also needed since in every node their attributes need to be updated. If new pokemons are not created we cannot access the correct values of attributes of the pokemons at any level but max level. Finally, after the creation of the graph, destructors are needed for both graph and node classes to avoid memory leaks. Destructors follow a BFS algorithm too, by visiting every node it deletes the pokemons than the node itself.
The second function is *part1* which prints every node in the specified level by traversing the graph again using BFS.

2.2 Part 2

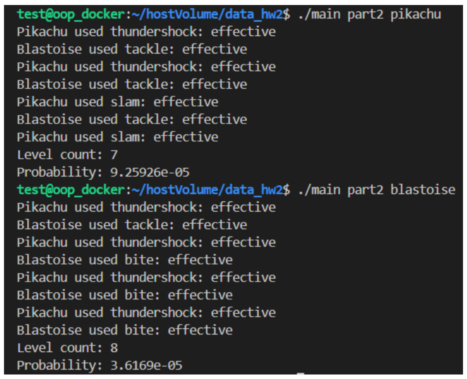
Part 2 required finding the shortest path to win the battle for both pokemons. This time the BFS search is used to find the shortest path rather than traversing the whole graph. I added a parent attribute to the node class since it was easier to trace back after finding the specific node. To print the path as expected in the provided file of the assignment, I implemented a stack and added the parents of the node until the root. Lastly, the status of the elements in the stack is printed.

3 Results



```
test@oop_docker:~/hostVolume/data_hw2$ ./main part1 0
P_HP:200 P_PP:100 B_HP:200 B_PP:100 PROB:1
test@oop_docker:~/hostVolume/data_hw2$ ./main part1 1
P_HP:200 P_PP:90 B_HP:160 B_PP:100 PROB:0.333333
P_HP:200 P_PP:85 B_HP:150 B_PP:100 PROB:0.233333
P_HP:200 P_PP:85 B_HP:200 B_PP:100 PROB:0.1
P_HP:200 P_PP:80 B_HP:140 B_PP:100 PROB:0.266667
P_HP:200 P_PP:80 B_HP:200 B_PP:100 PROB:0.0666667
test@oop_docker:~/hostVolume/data_hw2$ ./main part1 2
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
```

Figure 1: Part 1 Results



```
test@oop_docker:~/hostVolume/data_hw2$ ./main part2 pikachu
Pikachu used thundershock: effective
Blastoise used tackle: effective
Pikachu used thundershock: effective
Blastoise used tackle: effective
Pikachu used slam: effective
Blastoise used tackle: effective
Pikachu used slam: effective
Level count: 7
Probability: 9.25926e-05
test@oop_docker:~/hostVolume/data_hw2$ ./main part2 blastoise
Pikachu used thundershock: effective
Blastoise used tackle: effective
Pikachu used thundershock: effective
Blastoise used bite: effective
Pikachu used thundershock: effective
Blastoise used bite: effective
Pikachu used thundershock: effective
Blastoise used bite: effective
Level count: 8
Probability: 3.6169e-05
```

Figure 2: Part 2 Results

4 Discussion

I needed to add a line as ”include cstddef” in the given doubly list header because I could not compile it due to an error ”NULL is not defined”. Also, for the second part, I found the first node that the chosen Pokemon wins rather than the one with the highest probability. If the highest probability is required, a list can be added to choose from. Finally, even if the chosen Pokemon is Blastoise, Pikachu is the Pokemon starting the game since it seems like that in the pdf file.