DOCUMENTATION AI MODULE

Oriol Manzano Duran
Rémy Bouvard
Hugo Lupa

Index

1-Class structure

The main classes used in the project are the following:

Initialize:The class used to start the communication with the server and initialize all the other classes required.

Problem:Abstract class that holds all the logics that are specific for each problem we solve, in our case le fantome de l'opera. Algorithm class will use it for, given a node, return all possible childs. It's in this class where the heuristic is calculated as well.

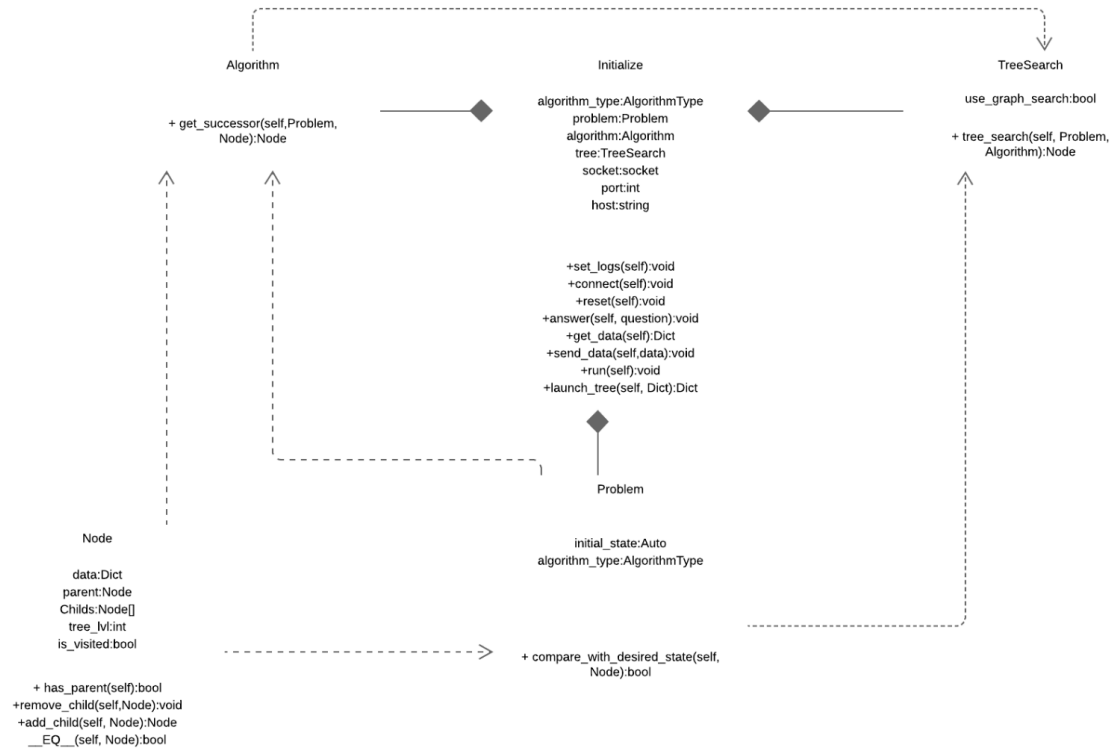
TreeSeach:Class containing the tree search loop that will in each iteration, first ask algorithm class, that given a node, return the best childs and then it will ask the problem class if any of those nodes can be a final acceptable solution or if it's necessary to continue in the tree search.

Algorithm:Abstract class with all logic of the algorithm. It's generic and can be used to solve any kind of problem. For instance in our project we have decided to use minmax

Node: Class with the necessary information to represent a node in the tree search

The project architecture is setup in a way that the algorithm logic is separated from the problem logic, meaning that the algorithm code can be used to solve multiple problem types. With this approach we can solve the "phantom de l'opera" problem with different algorithms alternatives without having to change the code of the problem.

The following image contains a uml diagram of the architecture



2-MinMax Algorithm

We have decided to use minmax algorithm for our project. But because there is not only one communication with the server in each turn, but rather multiple ones(”select character”, “select position”, activate power” etc), in our logic each node of the tree corresponds to a command sent to the server, meaning that instead of iterate between min and max in each tree level, we iterate between the set of nodes that makes a player turn.

In each iteration of the tree search loop we are going to call the get_successor of the minmax algorithm and its going to return a list of all nodes that give the best heuristic for the specific problem.

We ask minmax algorithm to give the moves to do only when server ask to select a character, because is when the turn starts and ask it for other commands like the power activation will be a waste of time, we replicate all possible player moves and enemy moves in the minmax so we know the set of player moves to do for the turn that will have a better heuristic.

3-Work Separation

Oriol Manzano Duran: Base class architecture, problem and algorithm
Remy Bouvard: Algorithm and problem.

4-Folder Structure

src/* Contains the base classes of the project

random_src/* contains logic for create a random ai, it's the same as the random_fantom and random_inspector given to us, but using our class architecture

minmax_src/* code specific to solve minmax problem
minmax_src/minmax_fantome.py File to execute the fantome
minmax_src/minmax_inspector.py File to execute te inspecotr
minmax_src/MinMaxAlgorithm.py Child class of Algorithm that contains information to solve the problem using minmax algorith
minmax_src/MinMaxProblem.py All the fantom_de_l'opera logic

5-Work still to do

We have to modify **minmax_src/MinMaxProblem.py** to correctly match exactly the same logic used in the server, right now is not working as expected.