Inteligência Artificial para Jogos

2017/2018

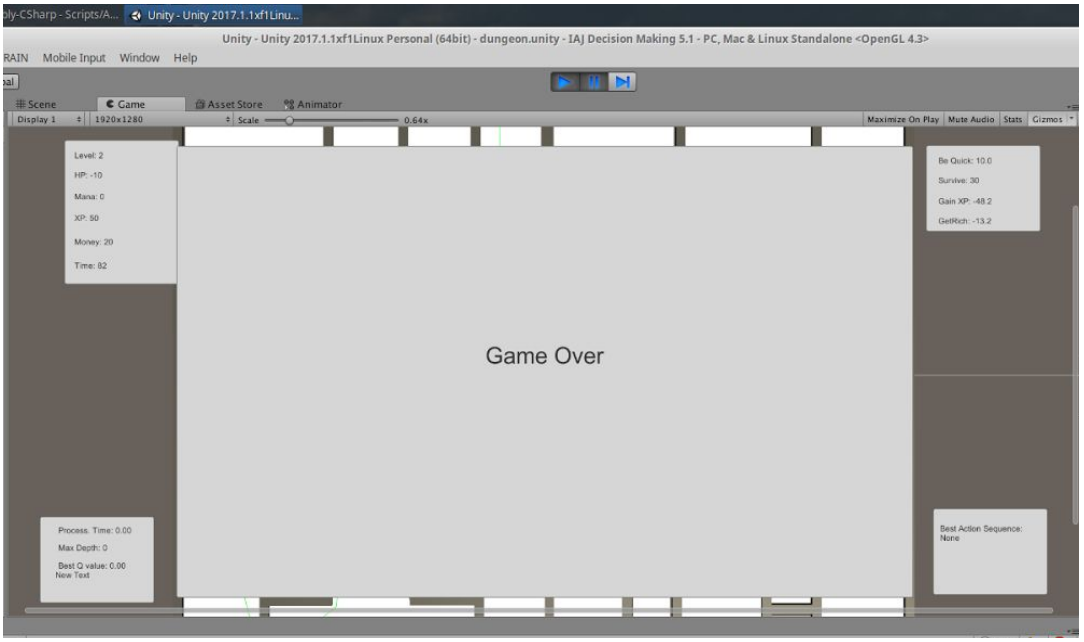
Relatório

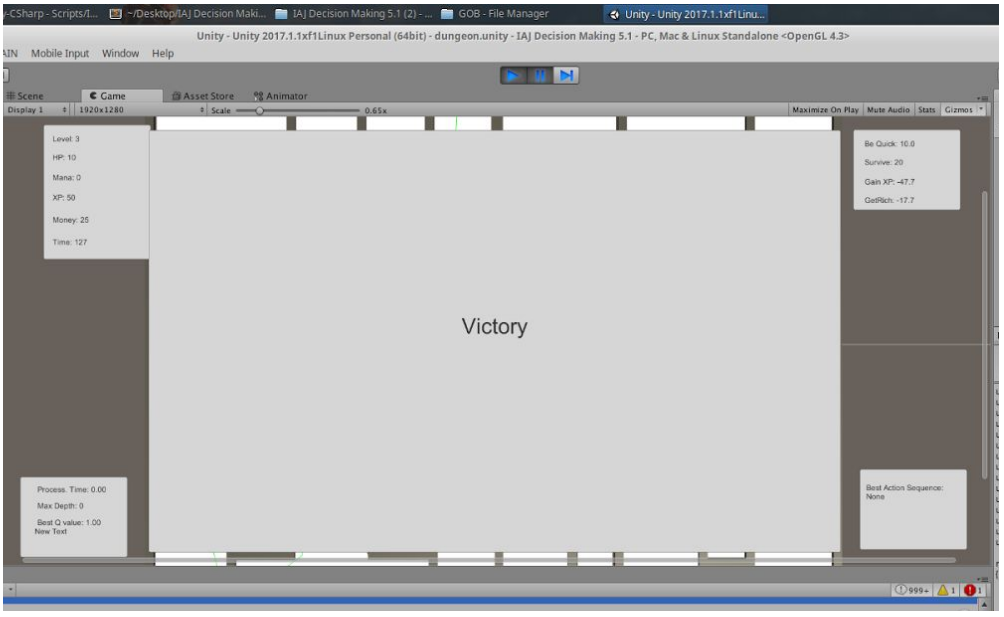
Projeto 3

André Vieira nº 79591
David Calhas nº 80980
Telma Barragão nº 89241

➔ Secret Level 1- Optimizing World State Representation

This kind of optimization was also implemented on the NodeArrayAStar algorithm, and the main idea is instead of using a List to record the values of the world properties, an array is used. As explained in the last project, using an array structure instead of a list, the access to the array is much faster than to the list, because the array is all together in memory as a block and in the list the values are dispersed in memory, with one position pointing to another, this means that if you want to iterate over the list, it is much more expensive than iterating over an array.





As we can see there was a better performance with the new world model.

➔ Secret Level 2- Comparison of MCTS variants

After implementing MCTS, following the lab guide (lab 7) document, we started implementing the MCTS Biased Playout variant, where the only difference between this variant and the normal MCTS is the playout implementation (in MCTS Biased Playout we use an heuristic).

One change that we decide to do was the Reward calculation. Before this, the calculation was returned by a if verification: if the money was 25 the reward value would be 1, else the value would be 0.

To change that, trying to get a most realistic value, we started to calculate the reward value with a money and a time percentage:

value = money / 25 - time / 200 * 0.15

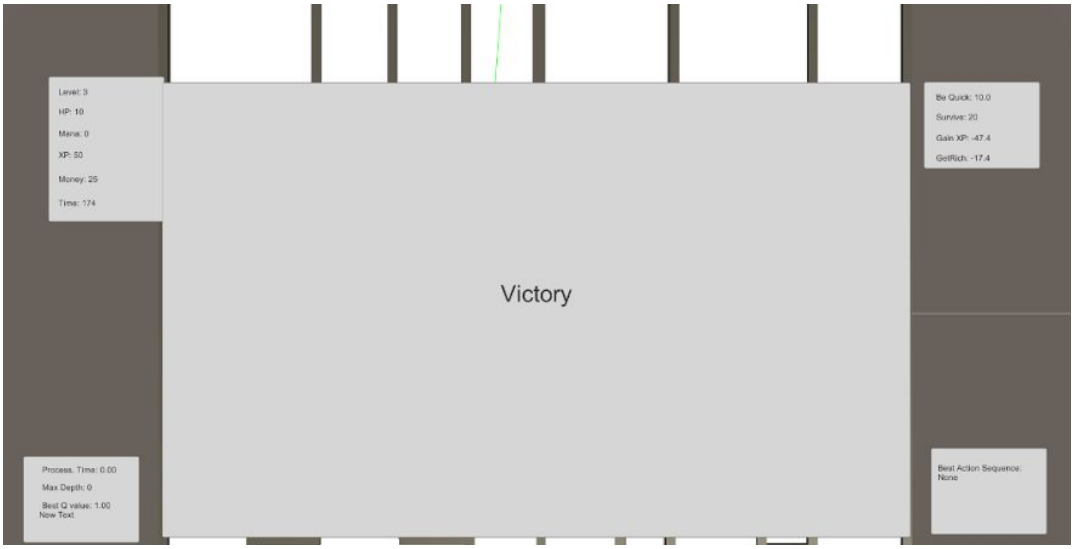
After that, when we were implementing MCTS Biased Playout, we decided to implement an heuristic based on the player goals values, meaning, for each action, in the executable actions list, after generating a new child world model and applying the actions effects on it, we calculate the heuristic value of that action, getting the player goals values of that world model.

Despite the fact that the player has 4 main goals (be quick, survive, gain xp and get rich), we decided to use all of them to increment the heuristic value, but the gain xp value we increment the inverted value, because when we applied the gain xp normal value to the heuristic calcule, the player used to decide to confront the dragon at his first action, dying.

During this “for” cycle, we keep the best heuristic value and the action it selfs, so we can get the best values from it, returning the best action.

When we run the normal MCTS (A* nodesPerFrame = 500, MCTS maxIterations= 10000, MCTS iterationsPerFrame=50):

When we run the MCTS Biased Playout (A* nodesPerFrame = 500, MCTS maxIterations= 10000, MCTS iterationsPerFrame=50):



The win rate% (total = 5 times):

normal MCTS = 100%

MCTS Biased Playout = 100%

In terms of time, the normal MCTS is faster that the Biased Playout variant, since the normal has the random in the playout, which is faster than the calculations that the Biased Playout has to do.

Comparing those two variants, we understand that the Biased Playout variant is better because the playout doesn't return just a random action. So, when the simulation of that action is applied, we believe that simulation will get us to a great state. Besides that, when we calculate the reward of this action, the probabilities of that reward being low and not the best, is too low.

➔ Secret Level 3- Compare MCTS with DepthLimitedGOAP

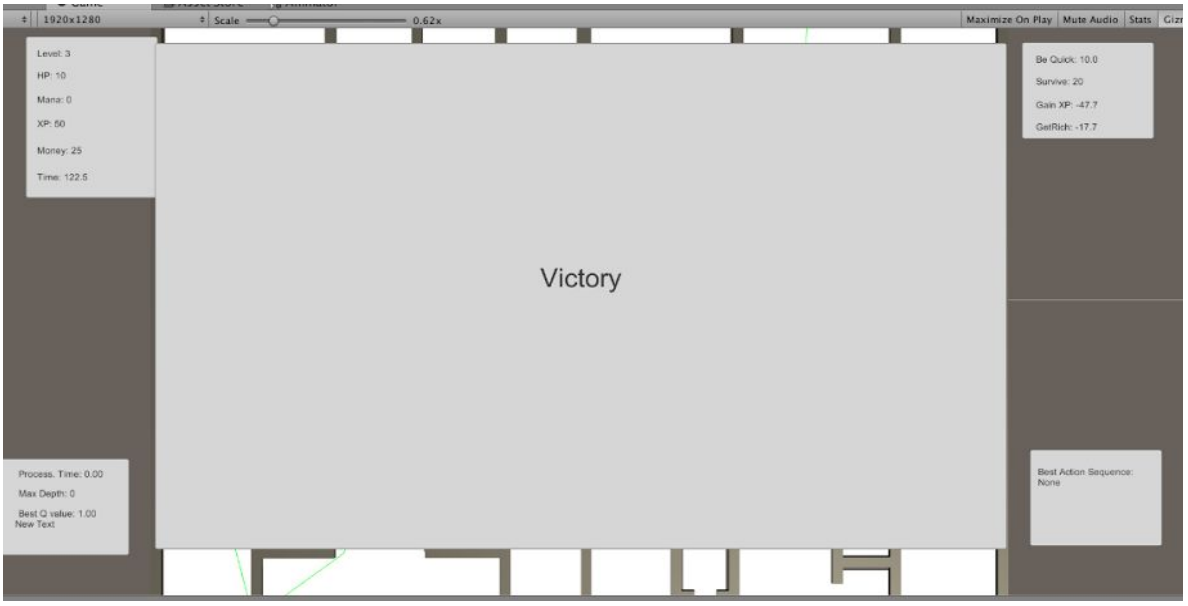
Comparing those two different algorithms, we understand that the MCTS is better than the DephtLimitedGOAP in some terms and DephtLimitedGOAP is better in some others.

First of all, in terms of time, the DepthLimitedGOAP is faster to make decisions than MCTS, because it has a certain Depth Limit and the MCTS has to simulate a lot of scenarios before it makes a decision.

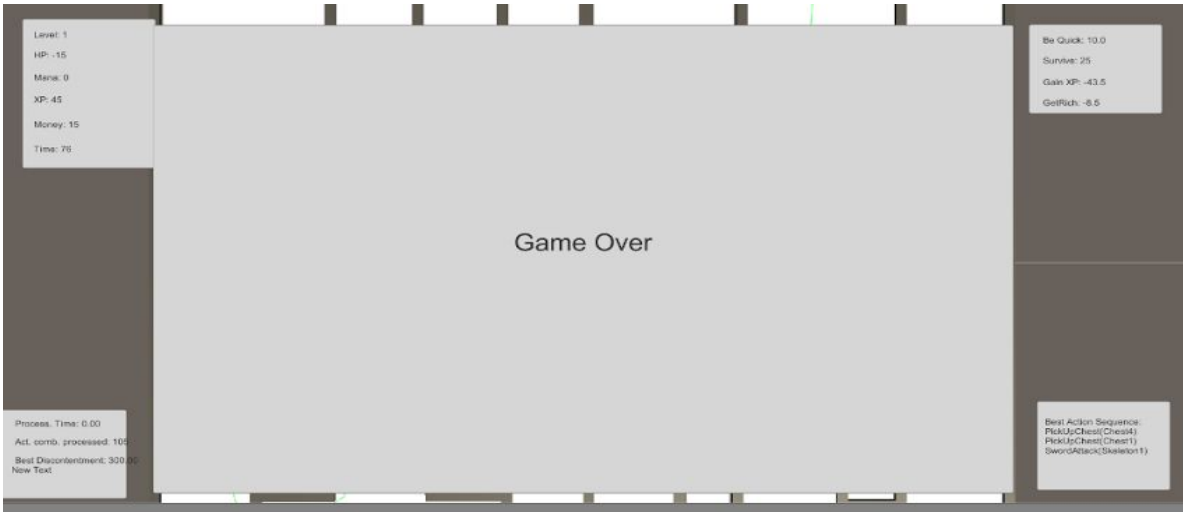
Second of all, we think that, in terms of making a good action decision it depends. This is because the MCTS algorithm is random in the playout, so it may not return a perfect solution, and the DepthLimitedGOAP returns a perfect solution with a determinated depth. So it really depends on the situation and the values that we are working with (MAX_DEPTH).

As the conclusion, we think that the MCTS is better than the DepthLimitedGOAP because he considers the whole world possibilities and the DepthLimitedGOAP only analyzes a certain limited depth of possibilities.

When we run the normal MCTS (A* nodesPerFrame = 500, MCTS maxIterations= 10000, MCTS iterationsPerFrame=50):



When we run the DepthLimitedGOAP (A* nodesPerFrame = 500, Max_Depth = 3):



The win rate% (total = 5 times):

normal MCTS = 100%

DepthLimitedGOAP = 0%

➔ Secret Level 4- Additional Optimizations

The first optimization that was made in our project, was to decrease the number of iterations of the MCTS search. We decreased it from 10000 as total and 500 iterations per frame, to 500 in total and 50 per frame. The results were really good, to the point where the slope that we could see in the Unity profiler did not appear anymore.

The second optimization, had to do with the MCTSBiased Playout, basically we just removed some operations that were not necessarily important for the game to have success. These operations were multiplications and variable assigns. In the examples, it shows that the execution was 3ms better than before.