Decision Making and MCTS

Ana Costa 81105, André Fernandes 78076 and Rodrigo Lousada 81115

December 2018

1 Depth Limited Goal Oriented Action Planning

The first approach was implemented due to the Laboratory 5. This approach allows an agent to plan a sequence of actions to satisfy a determined goal. GOAP can choose the best sequence of actions, evaluating all the action sequence with a maximum depth and applying a Depth First policy. Despite being effective, this algorithm faces a few problems. First, due to not having the Lab 7 implemented, the character only fetched chests. It toke this naive approach because it recalculated the action sequence after each action and the enemies did not attack the character.

Secondly, when the max depth is modified from 3 to 4, the character freezes. This is the result of the exponential growth of number of nodes that should be explored.

Lastly, due to calculating the distance between two points by Euclidean distance, the character would chose further paths instead of nearer ones. This overly optimistic calculation could be mended by calculating an estimate duration of a given path.

The actions implemented during this section were: DivineSmite, GetHealthPotion and ShieldOfFaith

2 Vanilla Monte Carlo Tree Search

In the second lab, Laboratory 6, we had to implement the algorithm Vanilla MCTS. The concept behind this method is starting from the root, advance in the tree by selecting a legal move and progress to the child node. The selection of the node is done by uning the folloing equation:

$$\frac{w_i}{s_i} + c\sqrt{\frac{\ln s_p}{s_i}} \tag{1}$$

Until a determined depth is reached, then we proceed to the expansion part. It chooses one random unvisited node and expands it. The next step it simulates randomly selected actions, until the end of the tree is reached (Playout). Lastly

the Backpropagation is done, after the simulation ends, the statistics on all the visited nodes are updated.

In this approach, by doing a random selection, on the Playout, the algorithm returns different answers. Yet, when we run MCTS enough times, it should return an answer close to optimal.

The problem that we faced when implementing this algorithm was the time interval that we gave between two actions. Th variable `DECISION_MAKING_INTERVAL` was initially set to 10 and we latter readjusted it to 100. This prevented the character from recalculating its path while in the middle of walking in a trajectory.

3 Stochastic environments, multiple players and Biased MCTS

The problem with applying MCTS in a stochastic environment is that there is always some level of randomness and the outcome can not be always predicted. To try to overcome this obstacle in a simple way we increased the number of times the playout performed, instead of doing it a single time. We tried to compute the playout 1,2 and 3 times. 3 times is to much, between 1 and 2 we the ratio velocity and quality. In the end we admitted that the best choice was to do the playout only once and return the reward with the max value. After applying this technique it was possible to observe that the agent acted in a more uniform way.

Now considering the Multiple Player method, after performing an action we considered if the character is close to an enemy. If it is, when we calculate a FutureStateWorldModel the next move will be from the enemy. If it happens on GameManager the character will have to fight back because the enemy will chase him until he do so.

Regarding the Biased MCTS we implemented a playout that influences the randomness to chose a given action, giving more probability to a set of actions based on a function CalcHeuristic. It each playout, a list is created that has the accumulated heuristic values. Then a number is generated, between 0 and the sum of the heuristic values. We see in what interval of the list that number falls into. By doing that, we choose the next action. To avoid some unwanted behaviours we gave a high heuristic to LevelUp or DevineWrath. By doing this the actions have more probability to be chosen. When calculating the heuristic, the algorithm analyses if the action is a WalkToTargetAndExecuteAction action. If it is not, we rise the heuristic to +12. On contrary, we give prior to the heuristic of nearest distance and consider the following cases: not attacking an enemy if the hp + shield is not high enough for the damage that the character can suffer and if the hp + shield are less than 30 percent, it favors the action of catching a health potion.

4 Optimizing World State Representation

In other to optimize the WorldState in a fixed size array, we changed the original WorldState into DictionaryWorldState which implemented an interface WorldState, after this we have created an ArrayWorldState.

	Avg. Processing Time	Avg. Actions Processed
DictionaryWorldModel	166	19394
ArrayWorldModel	137	18929

Figure 1: Difference between World State Representations

Comparing the both structures its possible to conclude that the Array reveals to be more efficient. Although the time, to the user, are almost unnoticeable. This is due to the efficiency of the dictionaries, they are only slow for recursive calls.

5 Limited Playout MCTS - Secret Level Two

The Limited Playout, similarly to GOAP, is applied to the Biased Payout with a max step of 5. This results on the brake of the While cycle. The cycle is waiting for the terminal state, with the possibility of reaching the max depth. This results on a shallow depth, due to the not needing to expand so many nodes. This restriction provides an improvement of the run time.

6 Comparison of MCTS variants

Comparing the regular MCTS with the Biased MCTS, we can observe that the Biased MCTS preforms better. This is due to the chosen heuristic.

Regarding the MCTS+BiasedPlayout and MCTS+Limited+Biased Playout, the conclusions are not that linear. The processing time of the second is much smaller. This is due to not executing the algorithm until reaching a final state. It only reaches a local optimal. However, MCTS+BiasedPlayout reaches a global optimal.

	Avg. Processing Time	Avg. Actions Processed	% win rate
MCTS	137	18929	0
MCTS+Biased	273	11120	0,2
MCTS+Biased+Limit	168	38732	0,1
MCTS+Biased+Limit+Opt	109	86604	0,6

Figure 2: Difference between MCTS Approaches

7 Additional Optimization. What else?

After realizing that we could not avoid catching a determined chest, specially if it was guarded by an enemy with the ability to kill the character, we crated a function called AvoidChestWithGard.

The function AvoidChestWithGard avoids choosing to pick up a chest if there is a monster guarding it. In the case of the Chest number 5, because the dragon is not right in front of the chest, if he is the last chest left to pick up, we chose to pick it up. This is possible because he can only touch the chest and win without facing the dragon.

Later, we also implemented a function that would prevent him to kill an Orc if he does not have enough hp to handle the damage caused by him. (On the 7th Lab, the damaged received decreased because of the random factor)

This solution exhibited good results, however two problems prevailed: he did not updated the level, what resulted in a wast of xp and health potions, and could not predict how close he was to each object, killing even a chest keeper without picking up the chest.

This lead us to associate each h value to each action and to sum that value to the exploration and exploitation factors on the BestUCTChild calc. The actions of LevelUp and DevineWrath received an h equal to 100 (trying to maximize the value). An to the WalkToTargetAndExecuteActions was assigned a value of 20 over the value of GetDuration (matching the Heuclidean Distance).