Rui Ribeiro
André Rodrigues
Frederico Ramos

Project 3
Group Number 3

2019/11/26

1 Depth Limited GOAP

As a first approach we used a Goal Oriented Behaviour(GOB) algorithm, specifically a Goal Oriented Action Planning(GOAP) derivation with limited depth. This algorithm considers instead of a single action, a sequence of actions that best meets the desired goals.

Depth Limited GOAP has a time complexity of O(MN^K) where M and N are the numbers of goals and actions and K is the max depth. In order to achieve better performance an optimization was implemented that prevents the algorithm from considering actions that increase discontentment, so if a world model's discontentment is greater than the one at the previous depth, simply ignore and backtrack.

This algorithm was tested on a non stochastic world with sleeping NPCs (Note that GOAP on non stochastic always produces the same result) with the following results:

Depth Limited GOAP	
Variant	Time to Win (s)
Not optimized	140,4
Optimized	69,2

We can clearly see that a lot of unnecessary actions were considered in the not optimized version (Ex:Attacking the Npcs even though they are not a threat). This situations are extremely reduced with the optimization.

When testing this algorithm with active NPCs, we observed that the not optimized version could not achieve a Win unlike the optimized one.

2 MCTS

After experimenting with GOAP we used the Monte-Carlo TreeSearch (MCTS) algorithm which is a generic algorithm for search problems.
The idea is to create a search tree by running a set number of simulations from the initial state until the end of the game, and then propagate the statistical information backwards.

2.1 Vanilla MCTS

The algorithm was adapted to a Stochastic environment by running the Playout() method a set number of times, in this case 5, to accommodate for the randomness by averaging the reward.
First we tested the the vanilla implementation of MCTS with fully random playouts and not limiting the depth of the search tree.

MCTS		
Win Ratio	Average Processing Time(s) (on Wins)	Average Iterations of MCTS.run() (on Wins)
50%	0,35	24

20 plays, Non-Stochastic World, 200 Iterations per frame

MCTS		
Win Ratio	Average Processing Time(s) (on Wins)	Average Iterations of MCTS.run() (on Wins)
20%	0,22	24

20 plays, Stochastic World, 200 Iterations per frame

MCTS		
Win Ratio	Average Processing Time(s) (on Wins)	Average Iterations of MCTS.run() (on Wins)
20%	0,41	22

20 plays, Stochastic World, 100 Iterations per frame

From the results above, we can see the vanilla MCTS performed average on a Non-Stochastic World and relatively poorly, with only 20% win rate, on a Stochastic World, remaining almost not influenced by the number of Iterations per frame.

2.2 MCTS Biased Playout

In order to improve on the vanilla MCTS, we can use an heuristic function to bias the random actions in the playout phase of the algorithm. Every action in the game has an heuristic function that tries to approximate how an action affects the current state of the world, in our implementation the better the action the lower the value will be.

Heuristic Functions:

1. Divine Smite:
   - Returns the Euclidean Distance between the character and the target GameObject divided by 25.
2. Divine Smite:
   - When the Mana value is over 5, returns the value of Walk to Target action divided by 5.
   - When the Mana value is under 2, returns 100.
   - In any other case return 15.
3. Get Health Potion:
   - When the character has more than 80% Max HP returns 100.
   - When the character has less than 20% Max HP returns 1.
   - When the sum of the character's health and shield is below 50% of the Max HP returns the value of Walk to Target action plus 5.
4. Get Mana Potion:
   - When the Mana value is under 1 returns 10.
   - When the Mana value is between 1 and 5, returns the value of Walk to Target action action plus 5.
   - In any other case returns 100.

5. Pick Up Chest:
   - If there are enemies close to the chest, returns the value of Walk to Target action divided by 7,5.
   - If there are none returns the same previous value, minus 50.
6. Rest:
   - When the character has more than 80% Max HP returns 85.
   - If still exists Health Potions available return 25.
   - When the character has less than 20% Max HP returns 0.
   - When the sum of the character's health and shield is below 50% returns 4.
   - In any other case returns the default value of 5.
7. Shield of Faith:
   - When the sum of the character's health and shield is below 60% and the shield is below 50%, returns 1.
   - If the shield is below 50%, and the mana is greater than 8, returns 2.
   - In any other case returns the default value of 20.

We chose a Gibbs distribution that gives a higher probability for lower heuristic values:

$$P(S, ai) = \frac{e^{-h(s,ai)}}{\sum_{j=1}^{A} e^{h(s,ai)}}$$

Biased MCTS		
Win Ratio	Average Processing Time(s) (on Wins)	Average Iterations of MCTS.run() (on Wins)
80%	0,07	22

20 plays, Non-Stochastic World, 200 Iterations per frame

Biased MCTS		
Win Ratio	Average Processing Time(s) (on Wins)	Average Iterations of MCTS.run() (on Wins)
30%	0,09	23

20 plays, Stochastic World, 200 Iterations per frame

From the results above, in a Non-Stochastic World, we can observe a big improvement from the vanilla MCTS, going from 20% to 80% win rate while reducing the processing time to about 1/5.

In a Stochastic World the performance improved to 30% win rate, having more 10% win rate comparing to Vannila MCTS.
Despite the heuristics' influence in the MCTS action selection, it does not prevent stochastic actions from enemies, and the win ratio could not improve as in a Non-Stochastic World.

2.3 Limited Playout MCTS

Since the improvement on the Biased MCTS was not the expected on the Stochastic World. We implemented a variation of the algorithm that limits the playout.
The goal is to overcome a possible low variance of some action's heuristics values, by computing a lighter playout.

Limited Biased MCTS		
Win Ratio	Average Processing Time(s) (on Wins)	Average Iterations of MCTS.run() (on Wins)
40%	0,07	22

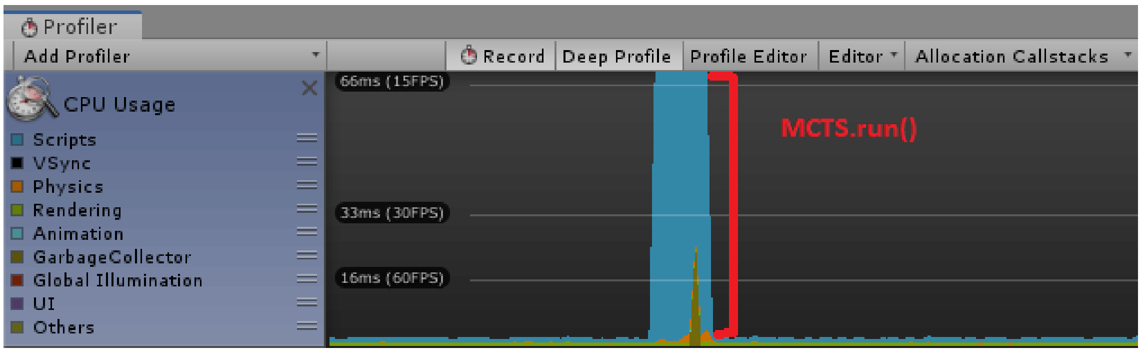
20 plays, Stochastic World, 20 Iterations per playout

We can observe a slight improvement in the win rate from 30% to 40%. This proves that limiting the depth of the playout could sometimes improve the win rate success, instead of considering all actions until the state is terminal.

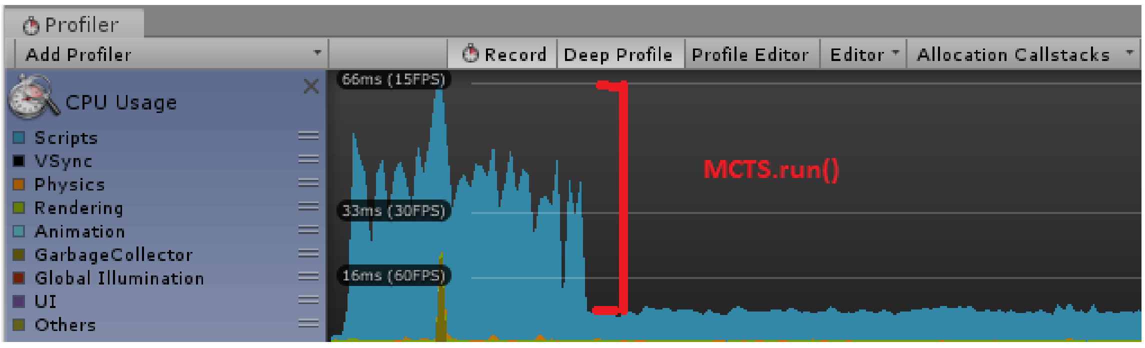
3 Optimizations

3.1 World State Representation

In order to optimize general performance, the World State representation can be optimized from being represented by a recursive model of dictionaries to a set of three arrays for the properties, enemies and resources, where every array position represents an instance of the respective object.



Before World State Representation



After World State Representation

3.2 Additional Optimization

The general MCTS algorithm can also be improved to prioritize instant actions and also the most important actions for the outcome of the game. In this particular case the Level Up action and Pick Up Chest action.
If in the first level of the search tree, prior to the execution of the algorithm, if any of both this actions is available, it can be immediately returned saving one full iteration of the MCTS.