IAJ Project 2 - Decision Making and MCTS

Group 6 :

- Vitor Vale nº 92570
- Tomás Saraiva nº 92564

Solution Description

Level 1 - Behavior Trees

To create the patrol behaviour for the orcs we started by making a new task which consisted in moving between targets in the map, being that every target was selected randomly from 6 targets in total, which introduced a bit of unpredictability into the gameplay.
Then we created a new behaviour tree for the orcs which had a selector type behaviour and it included the basic tree provided in the template and also contained the Patrol and Shout tasks.
The shout task consisted in making all the other orcs move to the last seen location of the player and after arriving if they didn't spot the player they would resume patrolling the map. We also added a sound effect as requested and a particle effect to mark the spotting of the player by the orc that shouted.

Level 2 - GOB and GOAP

For the GOB decision making algorithm we implemented the ChooseAction function as it was lectured in the classes and we also tweaked with the goal changes for each of the already implemented character actions. As a few examples of the changes we made for the get health potion action we used as a goal change the difference between the max hp and the hp at the moment when getting the potion, this way if the player is lower in health the decrease of discontentment will be greater by consuming the potion and therefore it will be more important to perform than other actions. Another example is the mana potion where we decided that if the character was in a short radius of the potion then it would be very beneficial to get the potion in order to perform other important actions that would also decrease the overall discontentment.

Level 3 - Sir Uthgard Actions

In order to correctly run the GOB and GOAP algorithms we implemented the get goal changes and apply action effects methods that were missing from the actions that were already on the template. We also added the actions that were missing such as Divine Smite and Shield of Faith for the GOB and GOAP algorithms and for the MCTS algorithm we also implemented the Rest and Teleport actions. To implement the methods mentioned we based ourselves on the level up and pick up chest actions that were already implemented. For the Divine Smite and Shield of Faith actions the rationale was that if there was mana available

the character would almost every time prefer to cast those abilities as soon as possible in order to get the most advantage before running into another enemy.
For the Teleport action the decrease of discontentment in using the action is as big as the number of enemies that are found near the character, so that way it would be more important to the character to escape to the starting position in order to avoid getting surrounded.
For the Rest action we made it so it was only beneficial to use if either the difference between the max hp and hp was very small or if there were no more health potions in the map and in that case the decrease of discontentment would be the same as the health potions, which was referred in the Level 2 description.

Level 4 - MCTS

For this algorithm and its variations the implementation we did followed the pseudo-algorithms described in the theoretical slides with almost no differences in terms of behaviour. We also implemented the multiple playouts for the stochastic environment as described in the lecture guide.

Secret Level 1 - Optimizing World State Representation

So to optimize the world state that was provided in the template we decided that instead of retrieving the properties that were not updated in the beginning by going through the parent states recursively we would include them all when starting the game and as the game went on they would be updated by the ApplyActionEffect's methods as they were before. We are still using dictionaries but now with a fixed size that corresponds to all the features relevant to the game state.
The goals were removed because in the normal MCTS algorithm the action chosen was random and because the variations of the later used instead heuristics to determine the best action. We also created a class Property and two others that inherited from the latter and they were used to contain the information relative to the features that were used in the algorithm.

Secret Level 2 - Limited Playout MCTS

For the Limited Playout MCTS algorithm the change to the implementation was simply to restrict the depth of the playout by a defined amount.

The heuristics implemented for the Biased MCTS algorithm to determine the best action were somewhat similar to how we determined the goal changes for the GOB and GOAP algorithms. The smaller the value attributed to the heuristic of a certain action the more it had a chance to be chosen as the action that was to be performed. As a few examples in the case of the pick up chest and get mana and health potion actions we took into account the distance that those objects were from the player and the more it was the more value was attributed to the heuristic. Also for the get health potion and rest actions we did a similar implementation in which if there were potions it would prefer to rest if the difference between max hp and hp was very low but if there were no potions the heuristic for resting would be the inverse of the difference between the max hp and hp, the same as the one for getting the health potion, because the heuristic had to be lower to be more important.

For some of the other actions such as leveling up or using shield of faith the heuristic was a very small value because we wanted the character to use them as soon as it could.

Secret Level 3 - Comparison of MCTS variants

Between the different variants we can see that the main difference is in the total processing time of each search. The variant that takes the longer to process is the Biased Playout MCTS due to the fact that some of the methods to get the H value have some calculations that takes longer than choosing a random best action like the standard MCTS does and also because it goes to max depth on contrary to the Limited Playout MCTS and because of that same reason the latter algorithm is the one that takes the least amount of time to run.

Secret Level 4 - Additional Optimization

We were unable to make any other additional optimizations.

Optimized World Representation in MCTS (Standard)

	Total Processing Time	Best Discontentment	Total Action Combinations Processed	Total Iterations Performed	Win Rate
GOAP	0.002	7.984	200		
MCTS (Standard)	0.04333333333			100	0
Biased Playout MCTS	0.05571428571			100	0
Limited + Biased Playout MCTS	0.03857142857			100	0

Standard World Representation in MCTS

	Total Processing Time	Total Iterations Performed	Win Rate
MCTS (Standard)	0.01785714286	100	0.3364285714

Final Remarks

On further analysis we concluded that the best algorithm for this type of gameplay is the GOB algorithm because it accounts better for unpredictability and also because it performs better when the gameplay is more dynamic instead of being an RTS type game.

Also we noted that our optimized world representation did not give better results in terms of processing time for the MCTS algorithm and we have an incline that the reason is because we waste too much time updating the features values, in theory though, having the values

present from the beginning should be beneficial in terms of time efficiency, because there is no need to go through the parent states to get values that weren't present since the beginning of the game, the downside would be the amount of memory used.

You won't be able to run the Biased and Limited Playout MCTS with the standard world representation, in order to do so you have to replace the class of the world representation in the MCTS and AutonomousCharacter classes.