Artificial Intelligence for Games
2nd Delivery - Group 12

Gonçalo Marques, 84719 Inês Vilhena, 84593 Pedro Maria, 84618

Introduction

This delivery consisted in developing several decision making algorithms to be used within the context of a video game, which included the G.O.A.P. (Goal Oriented Action Planning) algorithm and several M.C.T.S (Monte Carlo Tree Search) algorithm variations.

Lab 5 - Depth Limited GOAP

This algorithm was implemented by following the instructions given to us in lab 5's guide. It will simulate several combinations of actions from the initial state and then pick the best combination. The number of action combinations is limited by a depth limit.

3) a) The character's behavior was not corresponding to the expected behavior. Instead of going to objects that are closer to him, the character instead chooses to go to objects that are further away.

3) b) This happens because the character isn't prioritizing the "Be Quick" goal over other goals. This can be fixed by increasing the weight of the "Be Quick" goal and possibly decreasing the weight of the other goals.

3) c) With a depth of 3, GOAP processes around 10000 actions for each decision. If there was a depth of 4, around 200000 actions would be processed. If a depth of 4 is used, the character doesn't move because of the amount of different actions combinations being processed every time an action is chosen. A possible fix for this problem is limiting the amount of actions being processed.

Lab 6 - Vanilla MCTS

This algorithm was implemented by following the instructions given to us in lab 6's guide. It will create a tree out of World States and corresponding actions (Selection and Expansion stages) and then populate it with success values as it simulates those new states by applying random actions (Playout and Backpropagation). These values will then be used to determine which action is the best.

3) b) This was hard to confirm, but in theory, MCTS should make better decisions when performing more iterations over the same world state, given that it can generate more accurate reward values.

3) c) The character will choose worse actions with a 30 second limit because it cannot win in such a short amount of time, so the reward for all actions will be bad.

Secret Level 1 - Optimizing World State Representation

In order to optimize the existing World State representation we implemented the variant that contains an array with all the data that represents the world's properties (character stats, enemies and items) and another array with all the goal values.

Conclusion

After testing this new representation against the default implementation we concluded that the same amount of calls to functions like GetProperty() and GetGoalValue() were faster when using the new representation rather than the default implementation:

Table 1: World State vs Property Array World State

Method	Avg. execution time (ms) improvement
GetProperty	0.002484 –> 0.002326
SetProperty	0.001369 –> 0.000132
GetGoalValue	0.002866 –> 0.002416
SetGoalValue	0.001337 –> 0.001185

Secret Level 2 – Limited Playout MCTS

This variation of the MCTS algorithm involves limiting the depth that can be reached when executing a Playout while also biasing the selection of an action during that stage, as opposed to picking one randomly.

Our heuristic, that returns a value between 0 and 1, is as follows:

- If the action is LevelUp or DivineWrath, return a score of 1
- If the action is DivineSmite, return a score of 0.95
- If the action is ShieldOfFaith, return a score of 0.9
- If the character's hp is less than half:
  - If the action is LayOnHands, return a score of 1
  - If the action is GetHealthPotion, return a score between 0.7 and 1, depending on the distance of the potion
  - If the action is SwordAttack, return a score of 0.01
- If the character's mana is less than half:
  - If the action is GetManaPotion, return a score between 0.7 and 1, depending on the distance of the potion
- Otherwise return the amount of time passed (from 0 to 1)

Secret Level 3 – Comparison of MCTS variants

Note: none of our MCTS variants ever reached a victory, so the score we gave each one is based on our perception of the quality of the average run using that variant.

Table 2: Vanilla MCTS vs Biased MCTS vs Limited Biased MCTS

Property	MCTS	Biased MCTS	Limited Biased MCTS
Avg. Processing Time (ms)	215.11	598.91	488.32
Number of Iterations	200	200	200
Quality of Behavior	4/10	6/10	5/10

Conclusion

In terms of simply action choice quality, the Biased MCTS variant is the best option, but the best overall choice, considering both action choise quality and performance, is the Limited Biased MCTS, given that it can deliver relatively good results (within the limitations of the implementation) with reasonable performance.

Secret Level 4 – Efficiency optimization

The optimization we decided to do was the Move-Average Sampling Technique, where we associated a static Utility to each action and used it to bias the action selection during the Playout stage. This technique allows us to optimize the playout stage given that calculating the heuristic in unnecessary and a simple static value can be used instead.

We were able to reduce the average Biased MCTS processing time down to 270 ms as opposed to the almost 600 ms. This efficiency lead to a decrease in quality of action choice, but not as big as the leap in performance.