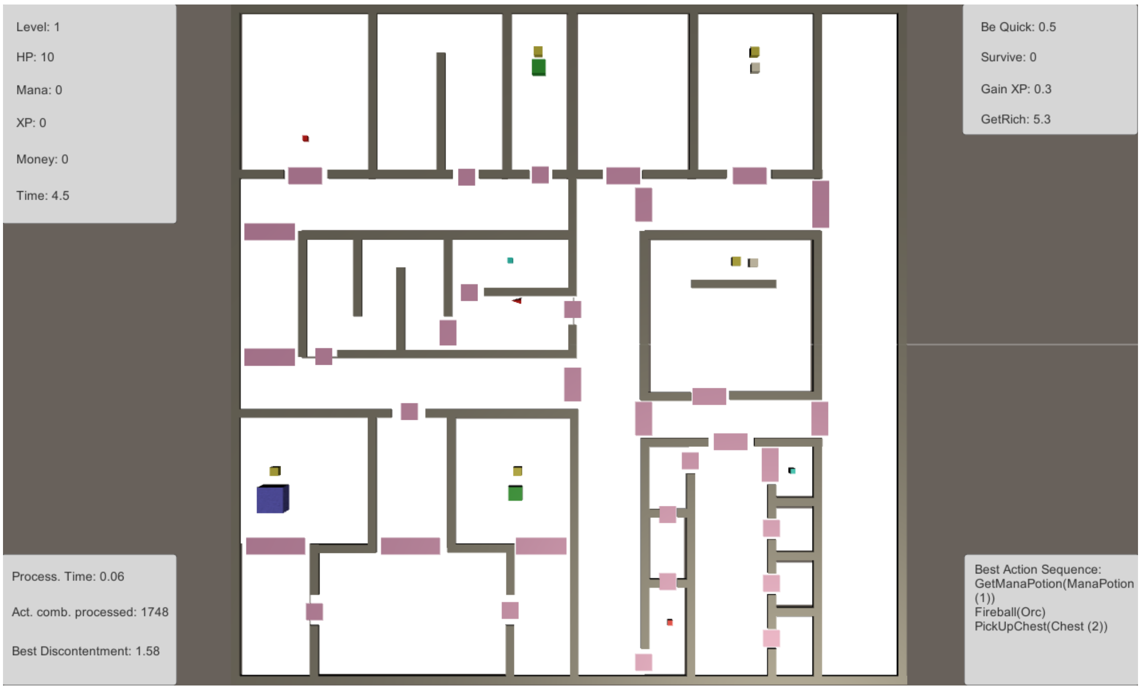
Inteligência Artificial para Jogos

2016-2017

Laboratory Guide

Lab 5 – Depth Limited Goal Oriented Action Planning



In this Laboratory we will explore the use of a Depth Limited GOAP (Goal Oriented Action Planning) algorithm, in order to consider multiple sequences of actions in order to select the best action to perform in a particular game scenario.

The scenario to be used in this Laboratory consists in a simple game environment, where the character has 5 variables:

- HP – represents the characters hit points.
- Mana – represents the character's mana, which is spent by performing fireballs. Getting a mana potion restores mana to its full value.
- Level – The current level of the character
- XP – The current experience level of the character.
- Money – how much money the character has collected
- Time – how much time elapsed since level started

There will be several distinct objects and enemies scattered across the level:

- Mana Potions – blue boxes, fully restore the character's mana (10 mana).
- Health Potions – red boxes, fully restore the character's health
- Chests – Golden boxes, represent the treasure chests that the character has to get. Each one gives 5 coins.
- Skeletons – Light grey boxes, the weakest type of enemy. They deal 2 damage to the character in melee combat. They also give 5 XP each.

- Orcs – Green boxes. Deal 5 damage to the character in melee combat. They give 10 xp.
- Dragon – Big blue box. Deal 10 damage to the character in melee combat. Hard to kill unless if you have mana for a fireball. It gives 15 XP.

In addition the character can perform the following actions in the environment:

- Sword Attack – attack a foe with your rusted sword. Can only be performed at melee range, and so monsters will attack you back. Attacking an enemy with your sword will decrease your HP according to their type as specified above. The enemy will be dead, and the character will gain XP accordingly.
- Fireball – Good old fireball. Ranged magical attack that can kill even the mightiest of enemies, if you have the mana for it. It uses 5 mana for each cast. The character won't lose any HP when attacking with a fireball because the enemies won't reach him before they are dead. The character gains XP accordingly.
- PickUpChest – Pick up a chest to give the character 5 gold coins.
- GetHealthPotion – Pick up a health potion and drink it, regaining full health equal to the character's maximum health.
- GetManaPotion – Pick up a mana potion and drink it, regaining full mana (10 points of mana).

1) Explore the provided scene and the source code

a) Open the downloaded project in Unity and explore the scene DecisionMaking. You will find the several distinct game objects scattered across the level.

b) Start by analyzing the classes inside the Game Manager folder. The GameManager class contains the methods for action execution and for updating the character's variables.

c) Integrate your code from project 2, into the AutonomousCharacter class.

d) Analyze the classes inside the DecisionMaking Folder. You will find two main classes, the WorldModel and the DepthLimitedGOAPDecisionMaking. The latter contains the code for the algorithm that you will need to implement. The WorldModel class is already implemented but it is very important for you to understand it. It represents the current or a future (simulated) state of the world. The current implementation uses a recursive WorldModel structure with dictionaries to store properties and goals. This is not the most efficient approach, but allows for more flexibility (i.e. easily adding new properties/goals).

e) Also take a look at the SwordAttack and PickUpChest actions, and the abstract WalkToTargetAndExecuteAction. The abstract action contains all the common functionalities to actions that involve moving before performing anything. All actions implement two CanExecute methods. The first one is the CanExecute that receives the WorldModel as argument. This method is similar to the CanExecute() without arguments, but instead of testing the game state to check if the action is possible to execute, it will test the received WorldState. The ApplyActionEffects assumes that the action was successfully executed in the received WorldModel, and therefore adds all predicted changes (in terms of goals and properties) to the WorldModel.

f) Also take a look at the class CurrentStateWorldModel. This class corresponds to a very particular case of a WorldModel where the WorldModel is actually the current game state. It serves as a kind of wrapper so that the current game state is handled by the GOAP algorithm in the same way as a WorldModel.

2) Implement the DepthLimited GOAP Decision Making

a) Implement the missing method in the DepthLimitedGOAPDecisionMaking class. Use the method GenerateChildWorldModel to perform a "copy" of a WorldModel. Use the GetNextAction to iterate over all possible actions for a WorldModel, and use the action's ApplyActionEffects to apply the results of an action to a newly generated WorldModel.

b) In addition to the standard algorithm, you will need to implement a mechanism that limits the number of action combinations (basically the number of actions that reach the last level) to be processed per frame (something very similar to what you did for the A* algorithm). Moreover, you will need to implement additional information that can be used for debugging, such as the TotalProcessingTime, the best action sequence (and not just the best first action), and the TotalActionCombinationsProcessed.
You will also need to implement the code for the Fireball and the GetHealthPotion action used in the Decision Making Process. Use the other implemented actions as examples.

3) Examine resulting behavior and experiment different parametrizations

a) Try out the resulting behavior and examine if the resulting behavior is according to the behavior you were expecting

b) It is easy to realize that very often the character will select objects to go to that are further away than other closer alternatives. Discuss with your group why does this happen, and try to figure a way how could you partially fix this.

c) You can experiment changing the defined value for the maximum depth. How many action combinations are processed by your algorithm at a maximum depth of 3? Can you roughly calculate how many will be at a maximum depth of 4? Experiment with a maximum level of 4. Why doesn't the character move? Discuss with your colleagues possible fixes to the problem.