Project MADI

Markov Decision Processes and Reinforcement Learning

Introduction

In this project we consider planning in a simple fantasy adventure game. In this game, an adventurer explores a dungeon (a 2 dimensional grid of fixed dimension) with the goal of getting a mythical treasure and coming back alive to the starting point. Each room of the dungeons is represented by cell of the grid. The game is turn-based; at each turn, the player moves according to the basic movement actions (going north, south, east and west) that are assumed to be deterministic. When the adventurer enters a cell, different things can happen depending on the "type" of the room (many of which have a random component). Furthermore the adventurer can collect some items (the golden key and the magic sword), in addition to the treasure.

The following table details the different types of cells:

Character code	Description	Effect
○	starting position	
(blank)	empty cell	nothing
■	wall	(an attempt to moving to a wall will bounce back to the starting position)
E	enemy	a malicious foe is attacking the adventurer (the adventurer is victorious with p_{enemy} probability, fixed to 0.7; otherwise the adventurer is dead).
R	trap	a trap that either kill the adventurer (with probability 0.1), bring the adventurer to the starting cell through an underground tunnel (probability 0.3) or nothing happens (probability 0.6)
C	cracks	immediate death
T	treasure	to open the door of the treasure's room it is necessary to have the golden key
S	magic sword	the adventurer can collect the sword and use it in combat; its powers allow him to win any combat without fighting
K	golden key	necessary to open the treasure's room
P	magic portal	a magic portal teleports the agent to a random (non-wall) cell of the dungeon
_	moving platform	the pavement is moving! the adventurer is forced to take refuge in one of the neighbouring cells (at random)

Note: there has to be a single treasure room in every dungeon (conveniently placed at the top-left corner of the grid, at the opposite of the starting cell) and there should be at least one room with the golden key and one with the sword. For example, the following is a possible 8x8 dungeon configuration (using the convention of the table above):

T	E			R	■	C	K
■	R	■	■	_	C	_	R
_		E				_	R
P	■		_	P	_	R	
S	■	■	E	■		■	■
_	R			R	_	C	P
	■	■	E	■		■	
	E						○

The game terminates either when the adventurer returns to the starting position with the treasure, or when he is killed in any way (by loosing a combat, by falling into a crack, etc). When the game restarts, obviously any objects found in the previous game is lost.

Part 1: Basic framework and MDP solver

Model the game as a Markov Decision Process by giving an appropriate large reward to the goal condition and by favouring winning the game in the fewer number of steps. Develop a software framework to represent and manipulate games of this kind. The software needs to have the following component:

- A framework to represents instances of the game as MDPs, with methods for random generation of games and methods for reading instances from a file. It is recommended to adopt an object-oriented approach for representing the different components of the problem. There should be a test for checking that a game is "winnable" (exists at least a path for getting the key and then the treasure and a way back).
- An interactive mode of playing the game where the input (the choices made by the adventurer) are given using the computer's keyboard and the current state of the game is displayed.
- A resolution component with at least two optimization methods (value iteration, policy iteration, linear programming) for computing the optimal policy.
- The possibility of visualizing the optimal policy and of simulating the game step by step using a given policy.
- At least one extra feature of your own choice (such as a more complex combat system, the availability of health points, new types of encounters, etc).

You should perform several tests with different game instances, including the example grid shown in the first page, and report the optimal policy for each of these. Make some additional tests varying the size of the board and the position of the rooms (tweak the probability parameters and choose a map so that winning is reasonably possible but not easy.), and report in a table (for the optimization methods that you have implemented) the computation times and the number of iterations.

Part 2: Reinforcement learning

The second part of project deals with model-free reinforcement learning. We now remove the assumption that the agent knows the map; moreover, the player ignores the transition probabilities (what happens in each room), but only observe the final outcome in terms of reward.

- Model the game as a model-free reinforcement learning problem.
- Extend the framework of part 1 so that it can support reinforcement learning.
- Implement the Q-learning algorithm to learn a policy.
- Train the Q-learning algorithm on the same dungeon maps that you considered in Part 1 and compare the obtained policies.
- Report on computation times for maps of different sizes.

Part 3: Markov games (optional part)

We now consider an extension of the game as 2-players game. Two opposing adventurers are present in the grid and compete for getting the treasure. The two players can interact in the following way: whenever one moves to the same cell as the opponent, the former can steal the treasure if it is in possession of the latter. The winner is the player who is able to get back the treasure to the starting cell.

Assume that a game's turn is executed as follows:

- Both players decide the next action to do
- Randomly pick one of the two players, execute his action, handle encounters in the landing room (including the possibility of stealing the treasure from the opponent, if applicable)
- Then do the same with the other player

Model the 2-players version of the game as a Markov Game and implement a strategy for learning how to play against an opponent (see reference below) and report results about matching different players with each other. Perform tests (in maps of relatively small size, so that encounters between the two players are not too infrequent) and describe which kind of policies can be found.

- Michael L. Littman: Markov Games as a Framework for Multi-Agent Reinforcement Learning. ICML 1994: 157-163

Instructions

- The project will be develop in `python`. Use only standard libraries and `numpy`.
- For solving linear programs, it is possible to use Gurobi (http://www.gurobi.com) that is available free of charge to students of Sorbonne université. Gurobi is available in the common computer science room. You may need to set the environment variables `GUROBI_HOME`, `PATH` and `LD_LIBRARY_PATH`
- The projects will be done in pairs of students. Each pair should send an email to paolo.viappiani@lip6.fr to declare themselves. (if you have not found a partner, contact the lecturer at the same email address). Include `[madi-project]` in the subject's email.
- The projects have to be sent to paolo.viappiani@lip6.fr by January, 13th. Use `[madi-project] submission` as the email's subject line. The submission will include a zip archive with the sources of the program, a `README.txt` file with instructions about how to execute the program, and a report (pdf).
- The report will describe the exact formulation of the game as MDP (how states, transition and rewards are represented), the algorithms implemented, the data structures (showing key points of the code). Motivate your implementation choices. The report will also describe experimentations for each of the parts (MDPs, reinforcement learning, and Markov games).
- The evaluation of the project will include a demo. Demos will take place on January, 24th, during the course hours.