Ex 2 Intro to AI
multi-agent search
AI warehouses

General instructions

- date of submission: **25.6.2023**
- The assignment must be submitted in pairs. special requests should be approved by the teaching assistant in charge (Sapir Tuble)
- The assignment must be submitted typed only - a handwritten solution will not be checked.
- The answers should be written in Hebrew or English.
- You can send questions about the exercise through the piazza.
- Teaching assistant in charge of this exercise: **Ofek Gottlieb**
- Justified late submissions should be sent only to the teaching assistant in charge (Sapir Tuble)
- During the exercise there may be updates - a message will be published accordingly in this case.
- The updates are mandatory and it is your responsibility to stay up to date on them by the time the exercise is submitted. Updates will appear in yellow in the form.
- Copying will be dealt with severely.
- The assignment score includes a dry part and a wet part. In the dry part we will check that your answer is correct, complete, legible and organized. In the wet part, your code will be comprehensively tested using automatic tests - which will compare your implementation with ours. It is important to carefully follow the instructions of the exercise since deviations from the requested implementation may lead to a failure in the automatic test (even if the implementation is "mostly correct"). During the automatic test, a reasonable time will be given for each run - so as long as you follow the instructions, your implementation will meet the time restrictions and return results good enough for the test.
- It is recommended to look at the code yourself. Basic questions about Python that do not pertain to the exercise should be checked online before you ask in Piazza. It is recommended to read the given code in order to understand how it works - in case there are things that are not understood. (For this purpose there are many comments and even an extended explanation about the environment!)
- It is recommended not to postpone the exercise to the last minute since the implementation and writing of the report may take longer than expected.

Submission instructions

inside a zip file with the name: HW2_AI_id1_id2.zip
the dry report in the format: id1_id2.pdf
and the file: subbmision.py (where you implement your algorithms)

Introduction

Bina warehouses want to make the warehouse autonomous and they are debating which robot they want to hire for the task. The robots Robot R1 and R2 must compete with each other and the winner will be hired.
In this exercise, you will implement and investigate adversarial zero-sum game algorithms that you learned in lectures and exercises.

Game description

The game takes place on a 5x5 grid board with: 2 robots, 2 charging stations, 2 packages and 2 destinations (one for each package).
Each robot has battery and credit units.

The goal of each robot is to score more points than the other robot until the end of the game, the game ends when one of the robots runs out of battery or when the maximum number of steps for each robot ends (predefined value, example below).

A score is earned when a robot brings a package to its destination. When a robot brings a package to its destination, it receives N scoring units where N is the Manhattan distance between the package's original location and the package's destination multiplied by 2. After a package reaches its destination, it and its destination disappear and a new destination and package appear in their place. That is, at any time there are exactly two packages and two targets on the board, one for each package.

Robots start with a charge and no package. Each action of the robot costs him one battery unit. Robot can move up, down, right, left. In addition, a robot can charge the battery at the charging station when it is in the slot of the charging station, it does this by converting all of its scoring units into charging units. A robot can pick

up and drop off a package. A robot picks up a package when it stands in the same slot as it and performs a collection action A robot downloads a package when it is at the destination and performs a download operation. (It is impossible to download a package collected in a slot other than the destination)
A total of 7 actions are possible for the robot at any time:
move north, move south, move east, move west, pick up, drop off, charge
A move to an illegal slot will not be allowed (an illegal slot is a slot outside the boundaries of the board or one where the other robot is located). It will not be possible to claim, assemble, download in illegal slots. Any agent can charge at any of the charging stations.

Working space, running and debugging

Unlike exercise 1, this time you will work with py files and not pynb notebooks. You are invited to work on the exercise in any suitable IDE that is convenient for you to work in Python, we recommend pycharm from the jetBrains company to which you have a subscription on behalf of the Technion.

The environment you will work with is implemented in the WarehouseEnv.py file, you are welcome to review it.

Agents are implemented in the Agent.py file, which agents you will implement inherit, it is recommended to look at the agents implemented in it to understand how they work with the environment.
It is recommended to look at the get_legal_operator and apply_operator functions to understand how you interface with the environment.
Note that in the submission file there is an agent called hardcoded, its purpose is to help you understand the environment.

Running

Running a game is done by the following command line that is run from the terminal:

python main.py greedy random -t 0.5 -s 1234 -c 200 --console_print --screen_print

- The first argument (in this case greedy) specifies the algorithm by which agent0 will play
- The second argument (in this case random) specifies the algorithm by which agent1 will play
- Time limit for the step t- gets a value that represents the maximum number of seconds for the step
- A seed for receiving a random value s- receives a value that helps generate an environment randomly, when the same seed value is passed it will generate the same environment
- The maximum number of steps for agent c-
- value console_print-- an optional flag, when passed a print is made to the console of the game that looks like this:

```
robot 0 chose move north
[R1][P0][  ][C1][C0]
[D1][  ][  ][  ][  ]
[R0][P1][  ][  ][  ]
[  ][  ][  ][  ][  ]
[  ][  ][  ][D0][  ]
robots:  [position:(0, 2) battery: 19 credit: 0 package: [None], position:(0, 0) battery: 20 credit: 0 package: [None]]
packages on street:  [position:(1, 0) destination: (3, 4), position:(1, 2) destination: (0, 1), position:(0, 3) destination: (3, 3), position:(2, 3) destination: (0, 1)]
(1, 0) (3, 4) True
(1, 2) (0, 1) True
(0, 3) (3, 3) False
(2, 3) (0, 1) False
charge stations:  [position:(4, 0), position:(3, 0)]
```

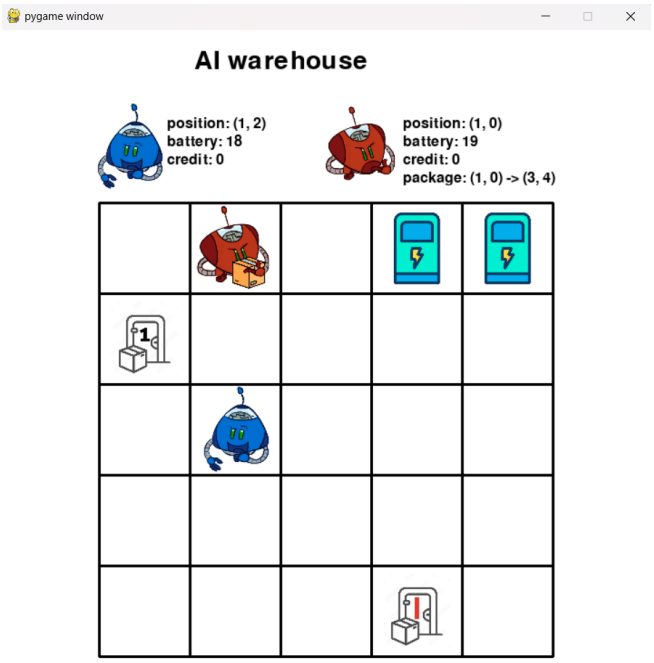
The agent number can be 0 or 1 and the operator you chose is printed. The board is then printed after the agent has run the operator. The board includes 25 slots where each can have a robot, a package waiting for the robot, or a charging station.
The letters that appear on the board symbolize:

R – Robot
C – Charge station
P – Package on street

D – Destination of Package on street
X – Destination of Package taken by Robot

-

- The number that appears next to each of the letters is the identifier of the object to which it belongs - for example: R1 for the first robot. For a package picked up by a robot, the number next to X is the ID of the robot that picked it up. For a package in space, the number next to D is the package's ID in space. Sometimes several objects appear in the same position in the image and then only one of them is printed. In this case, you can use lists the explicit ones that appear after the board. In addition, pay attention! The lists show you the source and destination of the next two packages that will appear on the board, you are invited and recommended to use this information later in the exercise.
-
- value screen_print-- an optional flag, when passed an animation of the game will be printed for you in the pygame window that looks like this:

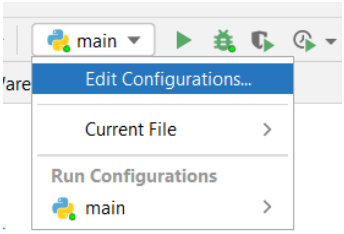


In addition to checking your agents on a large number of games, you can add the flag tournament-- and thus you will get a continuous run of n games. Feel free to play with the variable of the number of games and see if you get desired behaviors for the algorithms.

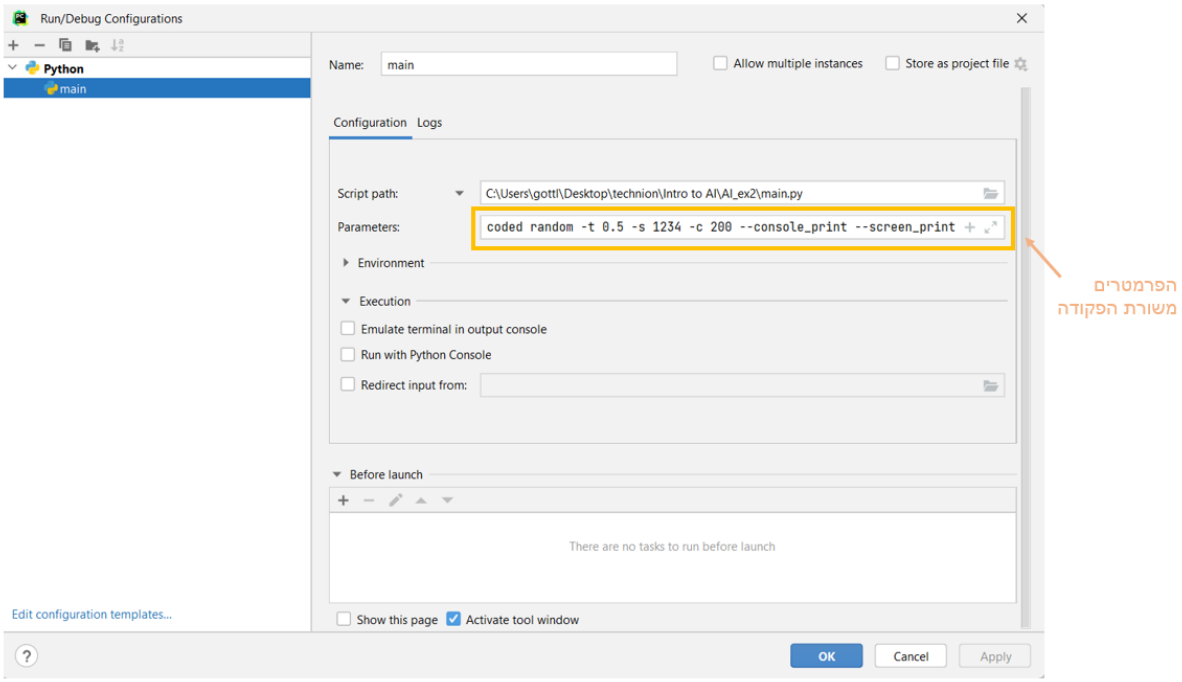
Debugging

To debug the game you must follow the following steps:

.1



.2



Let’s start writing!

Part A - ImprovedGreedy

1. (dry: 4 pt') As you have learned, we define a problem in space as a quadrant (S, O, I, G). Formally define the game described to you according to the data you receive from the environment.

2. (dry: 4 pt’) Define your own heuristics for evaluating game situations. You must document it in an explicit formula and include at least three characteristics of the environment. Choose clear names in your formula.

3. (wet: 10 pt') Implement the smart_heuristic function in the submission.py file which the AgentGreedyImproved agent uses

4. (dry: 2 pt') What is the main disadvantage of the algorithm? (compared to minimax)

Part B - RB-Minimax

1. (dry 3 pt') What are the advantages and disadvantages of using an easy-to-calculate heuristic versus a difficult-to-calculate heuristic given that the difficult-to-calculate heuristic is more informed than the easy-to-calculate heuristic? Given that we are in min-max limited resources.

2. (dry: 4 pt') Your classmate Dana implemented a minimax agent, she noticed that sometimes the agent can win with one move but chooses another move. Does it have a bug in the algorithm? If there is no bug, explain what in the algorithm causes such behavior. If there is a bug what could it be?

3. (wet: 10 pt') You must implement the AgentMinimax class in the submission.py file. pay attention! The agent is resource-limited, with the time_limit variable limiting the number of seconds the agent can run before returning a response. (The limit of the time you are being tested is one second, i.e. -t 1.

4. (dry: 3 pt') Let's say there were K players around instead of 2 (think of a general game or rather our game, but still a zero-sum game). What changes will need to be made in implementing the Minimax agent?
    a. Given that every agent wants to win and doesn't care only about you.
    b. Assuming that the only thing any agent wants is for you not to win.
    c. Assuming that each agent wants the agent next in line to win.

Part C - Alpha-Beta

1. (wet: 10 pt') The actual alpha-beta player has limited resources in the AgentAlphaBeta class in the submission.py file so pruning will be performed as learned in the lectures and exercises.

2. (dry: 3 pt') Will the agent you implemented in this part behave differently from the agent you implemented in part b in terms of running time and choice of moves? explain

Part D - Expectimax

1. (dry: 3 pt') Assuming you use Expectimax algorithms against an agent who plays completely randomly, what probability will you use? And why?

2. (dry: 4 pt') For probabilistic games like backgammon, where there is a resource limit, the RB-Expectimax algorithm is used. Assume that it is known that the heuristic function h in the Expectimax-RB algorithm holds $\forall s: -1 \leq h(s) \leq 1$
   How can this algorithm be pruned? Describe in detail the conditions of exaggeration, and explain the idea behind it.

3. (wet: 10 pt') The minimax and alpha-beta agent assumes that the adversary agent will choose the operator that will lead to the optimal outcome in turn, but this does not always occur.
   For example, when we compete with a greedy agent, it is likely that he will not choose the optimal action at every step. It is possible to consider the possibility that the opponent chooses a non-optimal action in his turn using the Expectimax agent.
   We will choose the weights of each operator in a way that takes into account that when the opponent can turn to a gas station, he will turn to it with a higher probability.
   If the agent can turn up, down, right, left, pick up a passenger, drop off, refuel and there is another gas station next to him, he is twice as likely to prefer the traffic towards a gas station. Literally a resource-limited ExpectiMax player that behaves as described.

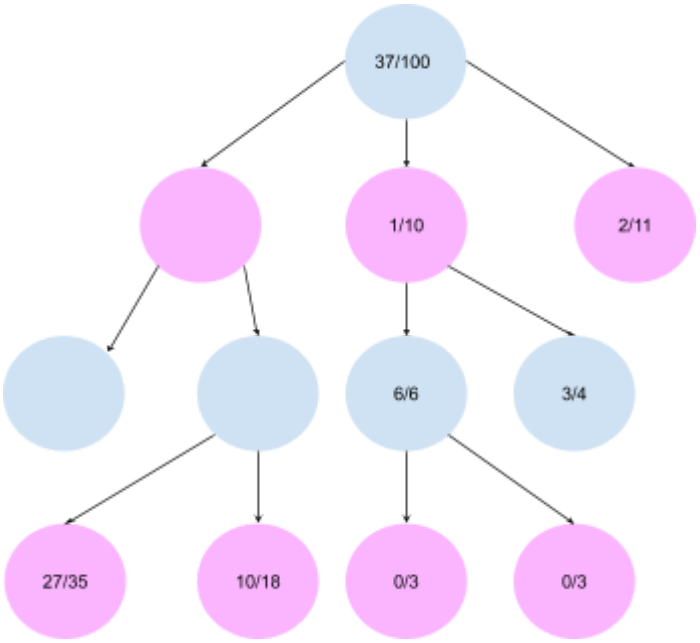
Part E - Games with large branching factor

1. (dry: 6 pt') Below are possible changes that Bina warehouses are considering to make in the game in order to test additional capabilities of the robots. For each change, state what its effect is on the creep coefficient and calculate the new creep coefficient obtained.
    a. Increasing the game board to be 8x8 and adding barriers in the environment. (Barriers mean slots that the agent cannot pass through)
    b. Adding the ability of a robot in each turn to choose a slot on the board and place a block, meaning that in each turn the robot can move up, down, right, left, pick up a package, drop a package, recharge, and place a block on the board, a block can be placed on any empty slot.

2. (dry: 6 points) Assuming that we implementing the second change for the environment (section 1b)
    a. Is there an algorithm from the previous sections that we can use that has a reasonable running time? (reasonable means not substantially greater than the time it takes him to return a step for the game without the change).
    b. Propose a different algorithm than the ones you implemented in the previous sections that will be taught in the course that will run in a reasonable amount of time. Explain why you chose it and why it is good for dealing with the challenge created by the changing environment.

Part F - Dry- Open question - MCTS

Dana and Tal played a game, Dana's turn is marked in blue and Tal's turn is marked in pink. Below is the game tree that describes the tree generated at an intermediate stage in the MCTS run with node development according to UCB1 on a zero-sum game between the two. Assume that the game has a maximum horizon of 10 steps, a given$C = \sqrt{2}$



1. Some of the values at the nodes have been deleted, fill in the missing parts, there is no need to explain.
2. Who is the next node to choose in the selection phase? Add calculations
3. Assuming that every simulation from here on ends with Tal's victory, and we define the most ancient ancestor to be the node closest to the root (a,b,c), what is the minimum number of victories necessary for a descendant of another most ancient ancestor to be chosen in the selection phase? (compared to section 2), add the relevant calculations
4. Now we want to make a change so that we prefer exploration over exploitation. Access to the formula that calculates the UCB1 is blocked for you, but the formula uses N(s) which you have access to and can change. How will you change it so that the new formula that will be created will favor exploration more than exploitation compared to the previous formula.