Assignment 3 - Minimax and Alpha-Beta Pruning

Morgan Heggland & Patrik Kjærran - MTIØT
TDT4136 - Introduction to Artificial Intelligence

October 2, 2019

1 Introduction

Assignment 3 in TDT4136 revolves around adversarial search applied to the classic game Pacman. The assignment asks us to implement the action deliberation for pacman using the minimax algorithm, and then improving the algorithm by use of alpha-beta pruning. These algorithms determine the ideal action for Pacman to take in a given game state, taking adversarial agents' ideal actions into account. In this case, the adversaries are the ghosts chasing Pacman. This document serves as an assignment report to complement and explain our solutions as they appear in the submitted code. In the following sections we will describe our approaches and designs to the different exercises as well as the results we obtained.

2 Deliverables

The only content delivered other than this report is `multiAgents.py`. Here, the minimax agent and and alpha-beta agent action deliberation is implemented.

3 Minimax implementation

Seeing as Pacman may oppose multiple adversaries, we have developed a generic minimax algorithm for n adversaries. The implementation is partly based on the pseudocode provided in the syllabus textbook [1]. However, in the textbook the algorithm is divided into three separate functions: `Minimax-Decision`, `Max-Value` and `Min-Value`. For the general case of n adversaries, we deemed it more appropriate to implement the algorithm as a single function. The psuedocode is based on alternating calls to `Min-Value` and `Max-Value` for each layer in the search tree. This is not the case for an arbitrary number of adversaries, as the maximizing agent is followed by an arbitrary number of minimizing agents.

In order to store information related to discovered game states in a convenient way, we implemented a `Node` class. Each instance of this class represents an encapsulation of explored game states. For the second part of this assignment, this also includes the `alpha` and `beta` parameters associated with each state.

The implementation is generic in the sense that it can be used in any situation where minimax may be used, as it takes all situation-specific functionality as inputs to the function. This includes:

- `node`: Node encapsulating the initial state to be evaluated
- `successor_gen`: Generator yielding all successor states for a given agent along with the action taken to generate each state
- `utility_func`: State evaluation function
- `terminal_func`: Function determining whether a given state is a terminal state
- `depth`: Depth of search tree
- `num_adversaries`: Number of adversaries, defaults to 1

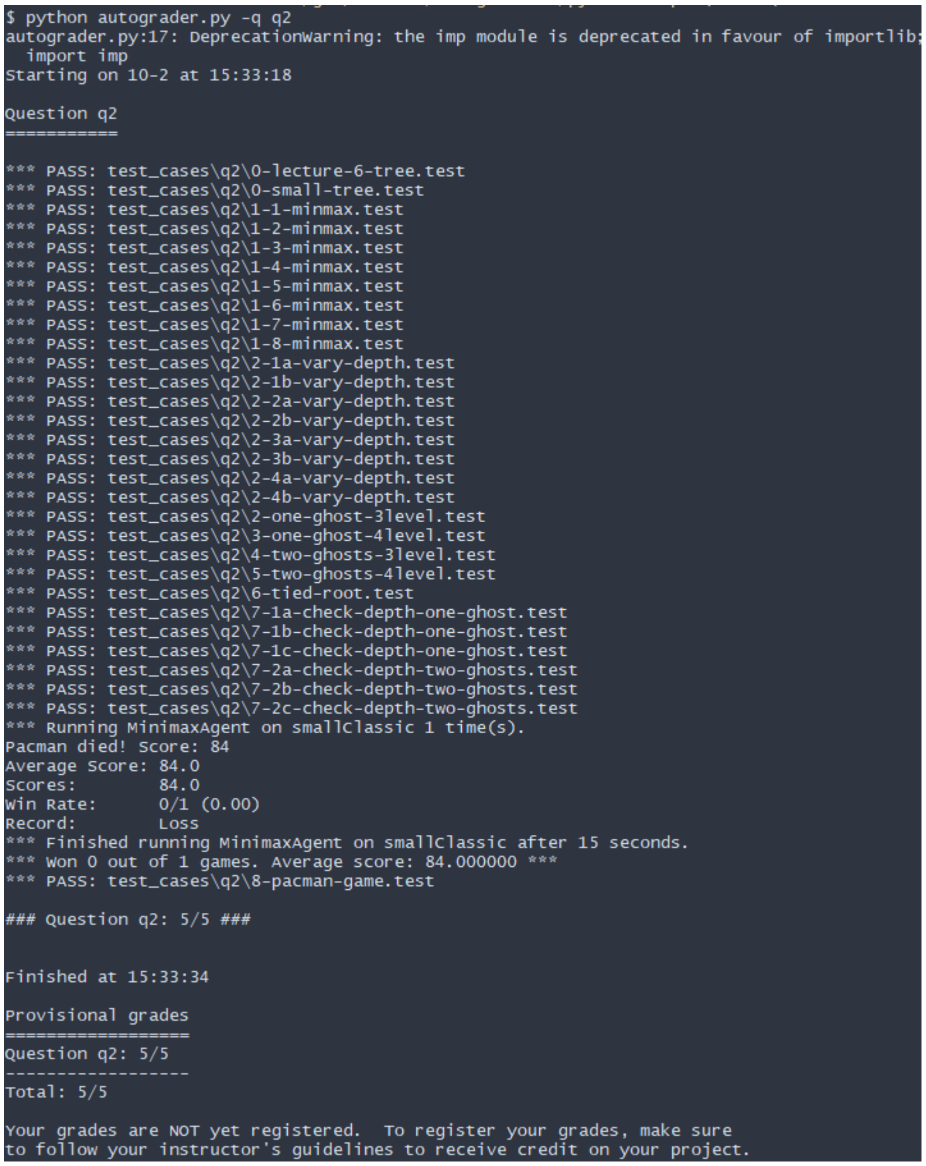
4 Question 2

4.1 Implementation

Question 2 asks us to implement a minimax agent for Pacman. Using the generic implementation described above, the remaining implementation consists of the Pacman-specific input functions. Given a game state, a node object encapsulating this state is generated. The `successor_gen` function takes in a game state and an agent index, determines the legal actions for said agent, and yields the generated successor states along with the action taken to generate it. `utility_func` simply reads the score attribute evaluated by the game logic for a given game state. `terminal_func` simply checks if the current state is a winning or losing state, returning false otherwise. `depth` is decided by the user running the agent, and `num_adversaries` is the number of ghosts in a given state.

4.2 Results

The implementation results in a Pacman agent able to survive quite well. We see that for a given depth, the agent does not progress in any meaningful way if all states around him are valued equally (no ghosts or food nearby). However, whenever there are ghosts or food nearby, the agent plays impressively well. In terms of runtime, we are able to run the agent with depth of 2 without significant time delay per action taken. The output from the autograder can be seen in Figure 1.



```
$ python autograder.py -q q2
autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib;
  import imp
Starting on 10-2 at 15:33:18

Question q2
===========

*** PASS: test_cases\q2\0-lecture-6-tree.test
*** PASS: test_cases\q2\0-small-tree.test
*** PASS: test_cases\q2\1-1-minmax.test
*** PASS: test_cases\q2\1-2-minmax.test
*** PASS: test_cases\q2\1-3-minmax.test
*** PASS: test_cases\q2\1-4-minmax.test
*** PASS: test_cases\q2\1-5-minmax.test
*** PASS: test_cases\q2\1-6-minmax.test
*** PASS: test_cases\q2\1-7-minmax.test
*** PASS: test_cases\q2\1-8-minmax.test
*** PASS: test_cases\q2\2-1a-vary-depth.test
*** PASS: test_cases\q2\2-1b-vary-depth.test
*** PASS: test_cases\q2\2-2a-vary-depth.test
*** PASS: test_cases\q2\2-2b-vary-depth.test
*** PASS: test_cases\q2\2-3a-vary-depth.test
*** PASS: test_cases\q2\2-3b-vary-depth.test
*** PASS: test_cases\q2\2-4a-vary-depth.test
*** PASS: test_cases\q2\2-4b-vary-depth.test
*** PASS: test_cases\q2\2-one-ghost-3level.test
*** PASS: test_cases\q2\3-one-ghost-4level.test
*** PASS: test_cases\q2\4-two-ghosts-3level.test
*** PASS: test_cases\q2\5-two-ghosts-4level.test
*** PASS: test_cases\q2\6-tied-root.test
*** PASS: test_cases\q2\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\q2\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\q2\7-2c-check-depth-two-ghosts.test
*** Running MinimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running MinimaxAgent on smallClassic after 15 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q2\8-pacman-game.test

### Question q2: 5/5 ###


Finished at 15:33:34

Provisional grades
==================
Question q2: 5/5
------------------
Total: 5/5

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

Figure 1: Output from the autograder for question 2.

5 Question 3

We are here asked to implement a variation of the minimax agent supporting alpha-beta pruning. Alpha-beta pruning is an extension to the minimax algorithm which seeks to decrease the number of evaluations in the search tree in order to improve runtime performance.

5.1 Implementation

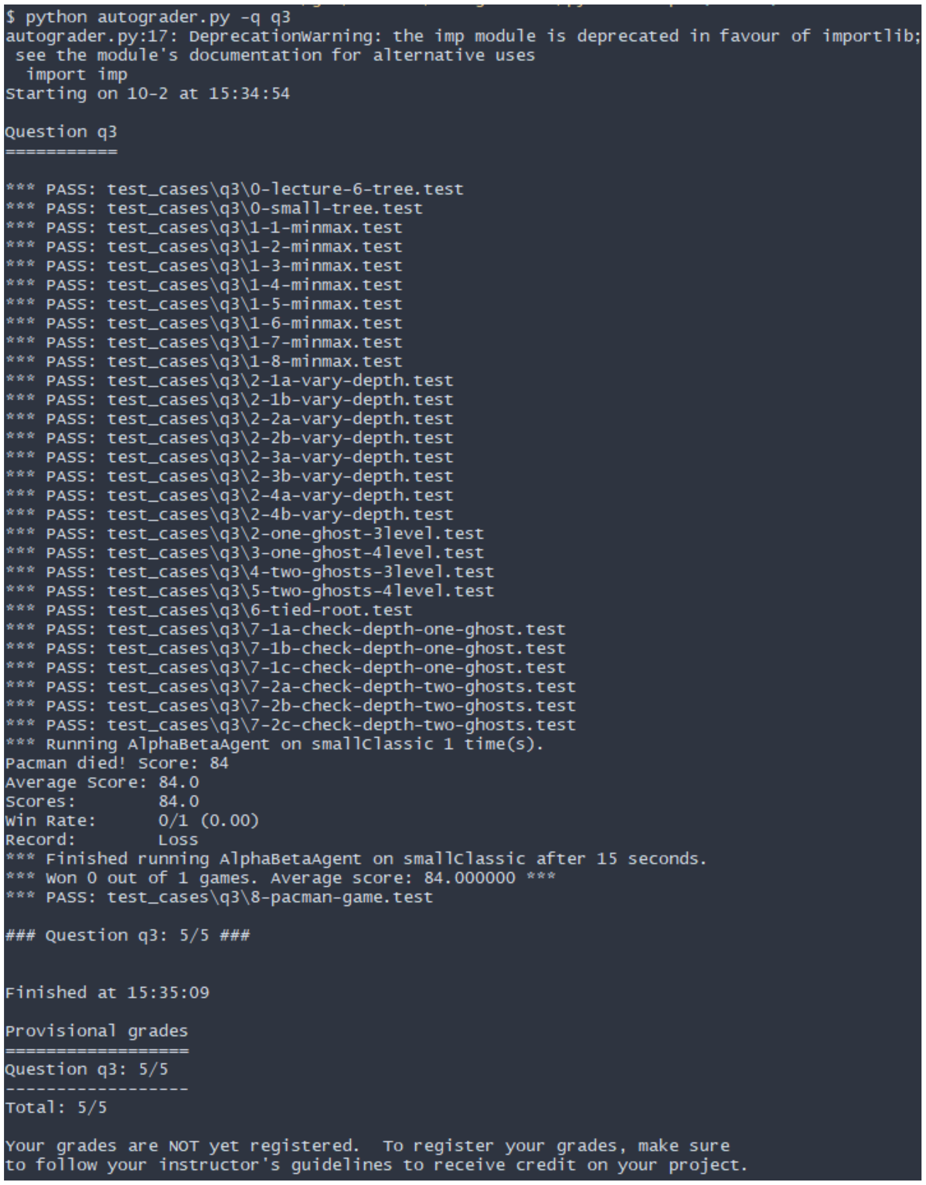
The implementation differs from the previous in two different ways. Firstly, the algorithm carries along two variables, `alpha` and `beta`, which represents the best value found in a maximizing and minimizing iteration respectively. It then uses these parameters when exploring new states to determine if any upcoming state expansions can be omitted.

5.2 Results

The difference in results of the alpha-beta agent compared to the minimax agent are purely related to runtime performance. With alpha-beta pruning, we are able to obtain the same performance running with a depth of 3 as the minimax agent did with a depth of 2. The output from the autograder can be seen in Figure 2.

References

[1] S. J. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2010.



```
$ python autograder.py -q q3
autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib;
 see the module's documentation for alternative uses
  import imp
Starting on 10-2 at 15:34:54

Question q3
===========

*** PASS: test_cases\q3\0-lecture-6-tree.test
*** PASS: test_cases\q3\0-small-tree.test
*** PASS: test_cases\q3\1-1-minmax.test
*** PASS: test_cases\q3\1-2-minmax.test
*** PASS: test_cases\q3\1-3-minmax.test
*** PASS: test_cases\q3\1-4-minmax.test
*** PASS: test_cases\q3\1-5-minmax.test
*** PASS: test_cases\q3\1-6-minmax.test
*** PASS: test_cases\q3\1-7-minmax.test
*** PASS: test_cases\q3\1-8-minmax.test
*** PASS: test_cases\q3\2-1a-vary-depth.test
*** PASS: test_cases\q3\2-1b-vary-depth.test
*** PASS: test_cases\q3\2-2a-vary-depth.test
*** PASS: test_cases\q3\2-2b-vary-depth.test
*** PASS: test_cases\q3\2-3a-vary-depth.test
*** PASS: test_cases\q3\2-3b-vary-depth.test
*** PASS: test_cases\q3\2-4a-vary-depth.test
*** PASS: test_cases\q3\2-4b-vary-depth.test
*** PASS: test_cases\q3\2-one-ghost-3level.test
*** PASS: test_cases\q3\3-one-ghost-4level.test
*** PASS: test_cases\q3\4-two-ghosts-3level.test
*** PASS: test_cases\q3\5-two-ghosts-4level.test
*** PASS: test_cases\q3\6-tied-root.test
*** PASS: test_cases\q3\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\q3\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\q3\7-2c-check-depth-two-ghosts.test
*** Running AlphaBetaAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running AlphaBetaAgent on smallClassic after 15 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q3\8-pacman-game.test

### Question q3: 5/5 ###


Finished at 15:35:09

Provisional grades
==================
Question q3: 5/5
------------------
Total: 5/5

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

Figure 2: Output from the autograder for question 3.