Report

October 28, 2017

1 Assignment 4 — Minimax and Alpha-Beta Pruning

Authors: Jakob Gerhard Martinussen and Andreas Friestad

1.1 About the Code

It should be noted that the entire assignment has been done in Python 3 instead of Python 2. This required porting the entirety of the code provided by Berkeley to Python 3. This has largely been successful, with one exception. Random number generation in python >= 3.2 behaves differently from earlier versions. This causes test `q3/8-pacman-game` to fail, as it depends on the old behaviour of `random`. This has been confirmed by running that specific test with Python 2 instead, where it did indeed pass. The error can be reproduced by passing the `-R` switch to the Python 2 interpreter, where an otherwise passing test will suddenly fail, because of hash randomization enabled by the `-R` switch. Otherwise all tests pass with Python 3.

The porting was assisted by the package `2to3` and the changes can be viewed here.

1.2 Question 2

The code for the question 2 minimax agent is as follows:

```
In [ ]: class MinimaxAgent(MultiAgentSearchAgent):
            """
              Your minimax agent (question 2)
            """

            def getAction(self, game_state: GameState) -> str:
                """
                  Returns the minimax action from the current gameState using
                  self.depth and self.evaluationFunction.

                  Here are some method calls that might be useful when implementing
                  minimax.

                  gameState.getLegalActions(agentIndex): Returns a list of legal
                  actions for an agent agentIndex=0 means Pacman, ghosts are >= 1

                  gameState.generateSuccessor(agentIndex, action): Returns the
```

```
        successor game state after an agent takes an action

        gameState.getNumAgents(): Returns the total number of agents in the
        game """
        legal_actions = game_state.getLegalActions(agentIndex=0)

        best_action_index = max(
            range(len(legal_actions)),
            key=lambda action_num: self.min_value(
                state=game_state.generateSuccessor(
                    agentIndex=0,
                    action=legal_actions[action_num],
                ),
                depth=self.depth,
                ghost_num=1,
            )
        )
        return legal_actions[best_action_index]

    def max_value(
            self,
            state: GameState,
            depth: int,
            actor: Optional[int] = None
    ) -> int:
        # Sanity check: have all the ghosts been evaluated the last round?
        if actor is not None:
            assert actor == state.getNumAgents()

        # Game over or search depth has been reached
        if state.isLose() or state.isWin() or depth <= 0:
            return self.evaluationFunction(state)

        legal_actions = state.getLegalActions(agentIndex=0)
        successors = [
            state.generateSuccessor(agentIndex=0, action=action)
            for action
            in legal_actions
        ]
        utilities = [
            self.min_value(state, depth, ghost_num=1)
            for state
            in successors
        ]

        return max(utilities)

    def min_value(self, state: GameState, depth: int, ghost_num: int) -> int:
```

```
        # Game over or search depth has been reached
        if state.isLose() or state.isWin() or depth <= 0:
            return self.evaluationFunction(state)

        # Sanity check: valid ghost number?
        assert 1 <= ghost_num < state.getNumAgents()

        legal_actions = state.getLegalActions(ghost_num)

        successors = [
            state.generateSuccessor(ghost_num, ghost_action)
            for ghost_action
            in legal_actions
        ]

        # If this is the last ghost, next optimizer should be from pacman's
        # perspective
        next_optimizer = self.max_value \
            if ghost_num == state.getNumAgents() - 1 \
            else self.min_value

        # If this is the last ghost, decrement depth
        next_depth = depth - 1 \
            if ghost_num == state.getNumAgents() - 1 \
            else depth

        utilities = [
            next_optimizer(state, next_depth, ghost_num + 1)
            for state
            in successors
        ]

        return min(utilities)
```

The commit of interest can be found here. The test output is as follows:

```
In [2]: %run autograder.py -q q2

Starting on 10-28 at 18:44:24

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 19 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 18:44:44

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.
```

1.3 Question 3

The code for the question 3 alpha-beta agent is as follows:

```
In [ ]: class AlphaBetaAgent(MultiAgentSearchAgent):
            """
              Your minimax agent with alpha-beta pruning (question 3)
            """

            def getAction(self, game_state: GameState) -> str:
                """
                  Returns the minimax action from the current gameState using
                  self.depth and self.evaluationFunction.

                  Here are some method calls that might be useful when implementing
                  minimax.

                  gameState.getLegalActions(agentIndex): Returns a list of legal
                  actions for an agent agentIndex=0 means Pacman, ghosts are >= 1

                  gameState.generateSuccessor(agentIndex, action): Returns the
                  successor game state after an agent takes an action

                  gameState.getNumAgents(): Returns the total number of agents in the
                  game
                """
                legal_actions = game_state.getLegalActions(agentIndex=0)

                alpha, beta = -inf, inf
                utility = -inf

                for action_num in range(len(legal_actions)):
                    successor = game_state.generateSuccessor(
                        agentIndex=0,
                        action=legal_actions[action_num],
                    )
                    utility = max(
                        utility,
                        self.min_value(
                            successor,
                            depth=self.depth,
                            alpha=alpha,
                            beta=beta,
                            ghost_num=1,
                        ),
```

```
            )

            if utility > alpha:
                best_action_index = action_num
                alpha = utility

        return legal_actions[best_action_index]

    def max_value(
        self,
        state: GameState,
        depth: int,
        alpha: int,
        beta: int,
        actor: Optional[int] = None,
    ) -> int:
        # Sanity check: have all the ghosts been evaluated the last round?
        if actor is not None:
            assert actor == state.getNumAgents()

        # Game over or search depth has been reached
        if state.isLose() or state.isWin() or depth <= 0:
            return self.evaluationFunction(state)

        legal_actions = state.getLegalActions(agentIndex=0)

        utility = -inf
        for action in legal_actions:
            successor = state.generateSuccessor(agentIndex=0, action=action)
            utility = max(
                utility,
                self.min_value(successor, depth, alpha, beta, ghost_num=1),
            )

            if utility > beta:
                return utility

            alpha = max(alpha, utility)

        return utility

    def min_value(
        self,
        state: GameState,
        depth: int,
        alpha: int,
        beta: int,
        ghost_num: int,
```

```
        ) -> int:

            # Game over or search depth has been reached
            if state.isLose() or state.isWin() or depth <= 0:
                return self.evaluationFunction(state)

            # Sanity check: valid ghost number?
            assert 1 <= ghost_num < state.getNumAgents()

            legal_actions = state.getLegalActions(ghost_num)

            # If this is the last ghost, next optimizer should be from pacman's
            # perspective
            next_optimizer = self.max_value \
                if ghost_num == state.getNumAgents() - 1 \
                else self.min_value

            # If this is the last ghost, decrement depth
            next_depth = depth - 1 if ghost_num == state.getNumAgents() - 1 else depth

            utility = inf
            for action in legal_actions:
                successor = state.generateSuccessor(
                    agentIndex=ghost_num,
                    action=action,
                )
                utility = min(
                    utility,
                    next_optimizer(
                        successor,
                        next_depth,
                        alpha,
                        beta,
                        ghost_num + 1,
                    ),
                )

                if utility < alpha:
                    return utility

                beta = min(beta, utility)

            return utility
```

The test output is as follows:

```
In [4]: %run autograder.py -q q3

Starting on 10-28 at 18:47:27
```

```
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 16 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** FAIL: test_cases/q3/8-pacman-game.test
***     Bug: Wrong number of states expanded.
*** Tests failed.

### Question q3: 0/5 ###
```

```
Finished at 18:47:44

Provisional grades
==================
Question q3: 0/5
------------------
Total: 0/5

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

As you can see, the last test fails with Python 3. The solution is hard coded into `8-pacman-game.solution`, and because of the changed hash function of python >= 3.2, this is no longer the correct result.

1.4 Source Code

You can find the entirety of the source code on Github.