Miguel Arnaiz Montes
S. Xiao Fernández Marín
G2391

INGENIERÍA INFORMÁTICA
Escuela Politécnica Superior
Universidad Autónoma De Madrid

Assignment 2: Search.

Miguel Arnaiz Montes
Sofia Xiaofan Fernández Marín

Pair 4 Group 1391
05/04/2021

Miguel Arnaiz Montes
S. Xiao Fernández Marín
G2391

Contents

1. Documentation of minimax + alpha-beta pruning **3**

a. Implementation details 3

i. Which tests have been designed and applied to determine whether the implementation is correct? 3

ii. Design: Data structures selected, functional decomposition, etc. 3

iii. Implementation. 4

iv. Other relevant information. 6

b. Efficiency of alpha-beta pruning. 6

i. Complete description of the evaluation protocol. 6

ii. Tables in which times with and without pruning are reported. 6

2. Documentation of the design of the heuristic. **7**

a. Review of previous work on Reversi strategies, including references in APA format. 7

b. Description of the design process: 7

i. How was the design process planned and realized? 7

ii. Did you have a systematic procedure to evaluate the heuristics designed? 7

iii. Did you take advantage of strategies developed by others. If these are publicly available, provide references in APA format; otherwise, include the name of the person who provided the information and give proper credit of the contribution as “private communication”. 7

c. Description of the final heuristic submitted. 8

Miguel Arnaiz Montes
S. Xiao Fernández Marín
G2391

1. Documentation of minimax + alpha-beta pruning

a. Implementation details

i. Which tests have been designed and applied to determine whether the implementation is correct?

We know that it works because we have run the `demo_tournament.py` and the functions that are used there, take less time to be run than the ones using `minimaxstrategy(Strategy)`. We have this checked in the efficiency of the alpha beta pruning part. Here, we have also printed on each loop, the value alpha and beta had at each moment.

We have also run the `demo_tictactoe.py`. Here we have changed the MinimaxStrategy to MinimaxAlphaBetaStrategy. We have used the verbose=2 because it permitted us to check the values of alpha and beta in each moment of the exploration.

We have checked the values by hand, and they were correct.

ii. Design: Data structures selected, functional decomposition, etc.

The data structure that is used for resolving this part of the assignment is a tree. This is because it is a minimax algorithm. It takes the advantage that some branches of the tree can be eliminated so the search time in this function is less than the minimax algorithm.

For resolving this, we have created a function called `next_move` that returns the next move the player must do. We have also implemented the function `max_value` and `min_value`, that are used by the `next_move` function. Those functions calculate the lower bound for the MINIMAX value at a MAX or the upper bound at a MIN.

What we are doing in `max_value` is, for each successor of the state we are in, we are taking the maximum between -∞ (just the first time we enter the function) with the one `min_value` returned to us. After that, if the value we have obtained is larger than beta, we return that value. If not, we check which value is larger, if the alpha value we had or the one obtained earlier. We choose the biggest one and we set it equal to alpha. Finally, we return the value obtained.

In the case we have reached a terminal state, we return the heuristic value of that state.

In the min value we have done the same but with alpha and beta reversed. We have also set the initial value to +∞. This means that for each successor, we compare the value of each initial value to the one returned by `max_value` and pick the minor that will be called `minimax_value`. After that we will compare it to the alpha value, that if it is lesser, we will return `minimax_value`. If not, we will check if it is lesser than beta, if it is, beta will be equal to that value. Finally, we return the `minimax_value`. In the case we have reached a terminal state, we return the heuristic value of that state, as we did in the `max_value` function.

iii. Implementation.

```
class MinimaxAlphaBetaStrategy(Strategy):
    def __init__(
        self,
        heuristic: Heuristic,
        max_depth_minimax: int,
        verbose: int = 0,
    ) -> None:
        super().__init__(verbose)
        self.heuristic = heuristic
        self.max_depth_minimax = max_depth_minimax

    def next_move(
        self,
        state: TwoPlayerGameState,
        gui: bool = False,
    ) -> TwoPlayerGameState:
        successors = self.generate_successors(state)
        minimax_value = -np.inf
        for successor in successors:
            if self.verbose > 1:
                print('{}: {}'.format(state.board, minimax_value))
            successor_minimax_value = self._min_value(
                successor,
                self.max_depth_minimax,
                minimax_value,
                np.inf
            )
            if (successor_minimax_value > minimax_value):
                minimax_value = successor_minimax_value
                next_state = successor

        if self.verbose > 0:
            if self.verbose > 1:
                print('\nGame state before move:\n')
                print(state.board)
                print()
            print('Minimax value = {:.2g}'.format(minimax_value))

        return next_state

    def _max_value(
        self,
        state: TwoPlayerGameState,
        depth: int,
        alfa: int,
        beta: int,
    ) -> float:
        if state.end_of_game or depth == 0:
            minimax_value = self.heuristic.evaluate(state)

        else:
            minimax_value = -np.inf

            successors = self.generate_successors(state)
            for successor in successors:

                if self.verbose > 1:
                    print('{}: {}'.format(state.board, minimax_value))
```

```
            successor_minimax_value = self._min_value(
                successor, depth - 1, alfa, beta,
            )

            if (successor_minimax_value >  minimax_value):
                minimax_value = successor_minimax_value

            if minimax_value >= beta:
                return minimax_value

            if minimax_value > alfa:
                alfa = minimax_value

    if self.verbose > 1:
        print('{}: {}'.format(state.board, minimax_value))

    return minimax_value

def _min_value(
    self,
    state: TwoPlayerGameState,
    depth: int,
    alfa: int,
    beta: int,
) -> float:
    if state.end_of_game or depth == 0:
        minimax_value = self.heuristic.evaluate(state)

    else:
        minimax_value = np.inf

        successors = self.generate_successors(state)
        for successor in successors:

            if self.verbose > 1:
                print('{}: {}'.format(state.board, minimax_value))

            successor_minimax_value = self._max_value(
                successor, depth - 1, alfa, beta,
            )

            if (successor_minimax_value <  minimax_value):
                minimax_value = successor_minimax_value

            if minimax_value <= alfa:
                return minimax_value

            if minimax_value < beta:
                beta = minimax_value

    if self.verbose > 1:
        print('{}: {}'.format(state.board, minimax_value))

    return minimax_value
```

iv. Other relevant information.

To complete this part of the assignment, we have followed the pseudocode that was given to us in the pdf.

We have also read in web pages the way this algorithm works as well as reading and comprehending the theoretical slides.

b. Efficiency of alpha-beta pruning.

i. Complete description of the evaluation protocol.

We have checked that it works using the demo_tournament.py and demo_tictactoe.py.

For that, we changed in tournament.py the MinimaxStrategy for MinimaxAlphaBetaStrategy. This gave us that the first one took 152.1233050s to be completed and the second one 14.6698082.

For checking this function with demo_tictactoe.py we have changed the player strategy and put the variable verbose equals to 2 because it gave us the value of beta and alfa, which was what we needed.

We have monitored the values from an intermediate state so that we know that the value returned by the heuristic is the evaluation of a terminal state.

ii. Tables in which times with and without pruning are reported.

MinimaxStrategy

```
PS C:\Users\sofia\Desktop\Ing. Inf III\2 Cuatri\AI\Practicas\P2> python3 demo_tournament.py

The time difference is : 152.1233053

Results for tournament where each game is repeated 10=5x2 times, alternating colors for
each player
```

	total	opt1_dummy	opt2_random	opt3_heuristic	opt4_Futuristic Heuristic
opt1_dummy	7	---	7	0	0
opt2_random	8	3	---	3	2
opt3_heuristic	19	10	7	---	2
opt4_Futuristic Heuristic	26	10	8	8	---

MinimaxAlphaBetaStrategy

```
PS C:\Users\sofia\Desktop\Ing. Inf III\2 Cuatri\AI\Practicas\P2> python3 demo_tournament.py

The time difference is : 14.6698082

Results for tournament where each game is repeated 10=5x2 times, alternating colors for
each player
```

	total	opt1_dummy	opt2_random	opt3_heuristic	opt4_Futuristic Heuristic
opt1_dummy	4	---	4	0	0
opt2_random	7	6	---	0	1
opt3_heuristic	25	10	10	---	5
opt4_Futuristic Heuristic	24	10	9	9	---

Miguel Arnaiz Montes
S. Xiao Fernández Marín
G2391

2. Documentation of the design of the heuristic.

a. Review of previous work on Reversi strategies, including references in APA format.

[1] Shiffman (Director). (2019, December 11). *Coding challenge 154: Tic tac toe AI with minimax algorithm* [Video file]. Retrieved March 07, 2021, from https://www.youtube.com/watch?v=trKjYdBASyQ

[2] *How to win at othello: Corner & Edge Strategies* [Video file]. (2018, August 06). Retrieved March 24, 2021, from https://www.youtube.com/watch?v=SvxTrjvPrSY

[3] Morillo Arroyo, D., & Busquiel Sanz, C. (2008). *Juego de Inteligencia Artificial: Othello* [Scholarly project]. Retrieved March 24, 2021, from http://www.it.uc3m.es/jvillena/irc/practicas/07-08/Othello.pdf

[4] Jacopo Festa, S. D. (n.d.). *"IAgo Vs Othello": An artificial intelligence agent playing Reversi* (Vol. 1104, pp. 3-5, Tech.). doi:http://ceur-ws.org/Vol-1107/paper2.pdf

b. Description of the design process:

i. How was the design process planned and realized?

So after collecting some information about the best strategies on how to win reversi, we decided to start by playing a little with simple concepts, such as the current scores of players, and we continued improving those same concepts until we obtained a decent result at the demo_tournament.

ii. Did you have a systematic procedure to evaluate the heuristics designed?

Every heuristic was evaluated through the demo_tournament.py file after modifying it so that it used the reversi game. After that we only changed the different heuristics we created to see the results.

iii. Did you take advantage of strategies developed by others. If these are publicly available, provide references in APA format; otherwise, include the name of the person who provided the information and give proper credit of the contribution as "private communication".

We did take advantage of some of the strategies shown in the 4 references above.

Miguel Arnaiz Montes
S. Xiao Fernández Marín
G2391

c. Description of the final heuristic submitted.

“FuturisticHeuristic”

This heuristic is the simplest one. We use the difference between the current score of the players.

“CornerAdorner”

This heuristic predicts the next best move. We obtained the successors of the current state and checked which was the next move returning 100 points if it was a corner, 40 if it was a lateral, -50 if it was the surrounding of a corner and 10 for any other square.

“WeightBait”

This heuristic returns the difference of the scores of both players. However, each square counts differently. For corners we gave 100, for the laterals we gave 40, for the squares surrounding the corners we gave -50 and for any other square we gave 10. Although, for the squares surrounding the corners, we only gave a negative score to the surroundings of one corner if there was no piece in that corner. In the case there’s a piece in a corner, we gave 30 points if it’s of the same color and 20 if it didn’t. We added all and that is what we returned.