Middle East Technical University | Department of Computer Engineering

CENG 462

Artificial Intelligence

Fall '2022-2023

Homework 3

Due date: 11 December 2022, Sunday, 23:55

1 Objectives

This assignment aims to familiarize you with optimal decision-making in two-player games at the implementation level and provide hands-on experience with two algorithms (Minimax Algorithm, Alpha-Beta Search) for solving them.

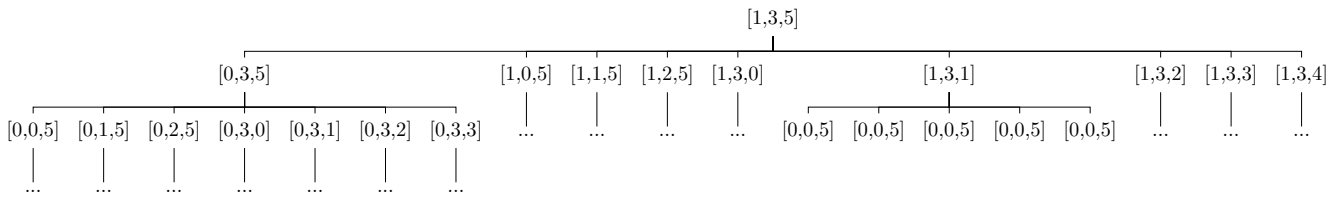
2 Problem Definition

Different from the search operations that were considered in the previous two assignments, where there was only a single decision-maker entity, two-player games involve two decision-makers that contend for defeating each other. In order to be able to make an optimal decision, both parties have to take their opponent's decisions into account during the search operation. Furthermore, if each decision-making step contains any element of chance, probabilistic inference has to be incorporated into the decision process.

In this assignment, you are expected to implement the Minimax algorithm and the Alpha-Beta Search method in order to solve the game of nim game and other generic two-player games that are represented as trees.

2.1 Game of Nim

Nim is a mathematical game of strategy in which two players take turns removing (or "nimming") objects from distinct piles.On each turn, a player must remove at least one object, and may remove any number of objects provided they all come from the same pile.Depending on the version being played, the goal of the game is to avoid taking the last object. In other words, The player that takes the last object loses the game. The number of piles and objects can be any size. [1,3,5] is example game. The number of piles is 3. The first pile has two objects, the second has three, and the third has three objects. The game tree is below:



The example game is [1,3,5]:

- Player 1 can choose one of the piles. Let's say he chose the first pile. Then,he took one object from the first pile. The game became [0,3,5]
- Player 2 chose third pile and he took one object. The game became [0,3,4].
- Player 1 chose third pile and he took three objects. The game became [0,3,1]
- Player 2 chose second pile and he took three objects. The game became [0,0,1] Player 2 win the game because Player 1 will take the last object.

If we assume the opponent at the root state is the max player. The terminal nodes (the nodes that the game can no longer progress, e.g. any player has won) have been labeled with the utility scores for the max-player. When a board configuration results in max's victory its utility value is 1. For a loss (min player wins), it gets a utility value of -1.

3 Specifications

- You are going to implement the Minimax algorithm and Alpha-Beta Search in Python 3.
- Each problem is represented with a file and this file will be fed to your implementation along with one of the methods.
- To be able to run all methods from a single place, you are expected to write the following function:

```
def SolveGame(method_name, problem_file_name, player_type):
```

  The parameter specifications are as follows:

  - `method_name`: specifies which method to be run for the given problem. It can be assigned to one of the values from `["Minimax", "AlphaBeta"]`.
  - `problem_file_name`: specifies the path of the problem file to be solved. File names are dependent on the problem and take values such as "nim1.txt", "nim2.txt"
  - `player_type`: specifies who is going to play in the game (whose turn it is now for the given game state, the player at the root state) (max player or min player) and takes the value of either "MAX" or "MIN".

  During the evaluation process, this function will be called and returned information will be inspected. The returned information is dependent on the method fed and as follows:

  - `method_name=="Minimax"`: `GameSolve` returns the optimal action, and the number of visited tree nodes during the search.
  - `method_name=="AlphaBeta"`: The same output structure as Minimax should be generated.

- For a particular configuration/state of the game of nim is provided in ".txt" files too and has the following structure:

```
[a,b,c,...]
```

  where a, b, c are numbers.

- For game of nim, while expanding a node (children construction) Let the game is (1,3,5), please follow the following action sequence: (0, 3, 5),(1, 0, 5),(1, 1, 5),(1, 2, 5),(1, 3, 0),(1, 3, 1),(1, 3, 2),(1, 3, 3)(1, 3, 4).

- For the nim games only the max-player (`X`, `player_type="MAX"`) is going to be tested.
- No `import` statement is allowed and all methods should be implemented in a single solution file. You are expected to implement all the necessary operations by yourself without utilizing any external library.
- Commenting is crucial for understanding your implementation and decisions made during the process. Your implementation can be manually inspected at random or when it does not satisfy the expected outputs.

4 Sample I/O

nim1.txt:

```
[1,3,5]
```

Expected outputs of GameSolve function with different methods applied on nim1.txt:

```
>>> import hw3solutioneXXXXXXX as hw3
>>> print(hw3.SolveGame("Minimax", "nim1.txt", "MAX"))
([(1, 3, 2), 446])
>>> print(hw3.SolveGame("AlphaBeta", "nim2.txt", "MAX"))
([(1, 3, 2), 110])
```

nim2.txt:

```
[1,2,4,5]
```

Expected outputs of GameSolve function with different methods applied on nim2.txt:

```
>>> import hw3solutioneXXXXXXX as hw3
>>> print(hw3.SolveGame("Minimax", "nim1.txt", "MAX"))
([(1, 0, 4, 5), 41992])
>>> print(hw3.SolveGame("AlphaBeta", "nim2.txt", "MAX"))
([(1, 0, 4, 5), 2575])
```

Note 1: There could be more one than one action that results in the same utility value, any of them can be returned as the picked action.).
Note 2: The problem files and their outputs here are provided as additional files on ODTUClass.
Note 3: When you return the number of processed nodes, you need to return the optimal possible move processed node. Let's say the game is [1,3,5]. The possible moves are (0, 3, 5), (1, 0, 5), (1, 1, 5), (1, 2, 5), (1, 3, 0), (1, 3, 1), (1, 3, 2), (1, 3, 3), (1, 3, 4). After this step, you need to go deep for every state. In other words, the state (0, 3, 5) has 1735 processed nodes, another state (1, 0, 5) has 143, and the state (1, 3, 2) is 446, and the same for other moves. We choose (1,3,2) as an optimal move, so we need to return the 446 instead of the total number of processed nodes. The numbers can be different. If the algorithm's logic is accurate and the same as the slides, there will be no penalty for different numbers.

5 Regulations

1. **Programming Language:** You must code your program in Python 3. Since there are multiple versions and each new version adds a new feature to the language, in order to concur on a specific version, please make sure that your implementation runs on İnek machines.

2. **Implementation:** You have to code your program by only using the functions in the standard module of python. Namely, you **cannot** import any module in your program.

3. **Late Submission:** No late submission is allowed. Since we have a strict policy on submissions of homework in order to be able to attend the final exam, please pay close attention to the deadlines.

4. **Cheating: We have zero-tolerance policy for cheating**. People involved in cheating (any kind of code sharing and codes taken from the internet included) will be punished according to the university regulations.

5. **Discussion:** You must follow ODTUClass for discussions and possible updates on a daily basis. If think that your question concerns everyone, please ask them on ODTUClass.

6. **Evaluation:** Your program will be evaluated automatically using the "black-box" technique so make sure to obey the specifications. A reasonable timeout will be applied according to the complexity of test cases. This is not about the code efficiency, its only purpose is avoiding infinite loops due to an erroneous code. In case your implementation does not conform to expected outputs for solutions, it will be inspected manually so commenting will be crucial for grading.

6 Submission

Submission will be done via OdtuClass system. You should upload a **single** python file named in the format **hw3_e<your-student-id>.py** (i.e. hw3_e1234567.py).

7 References

- For `Minimax Algorithm` and `Alpha-Beta Search`, you can refer to the lecture's textbook.
- Announcements Page
- Discussions Page