Introduction to Artificial Intelligence

Exercise 2: Two-player deterministic games

Valeriya Khan

Summer 2025

1 Task

Variant 1

Using the template from the files folder (lab2_v1.py), write a program that plays a **tic-tac-toe** game with the user using **min-max algorithm with alpha-beta pruning**. The board of game is a **3x3 matrix** with entries: " " (space) – empty cell, "X" – first player, "O" – second player.

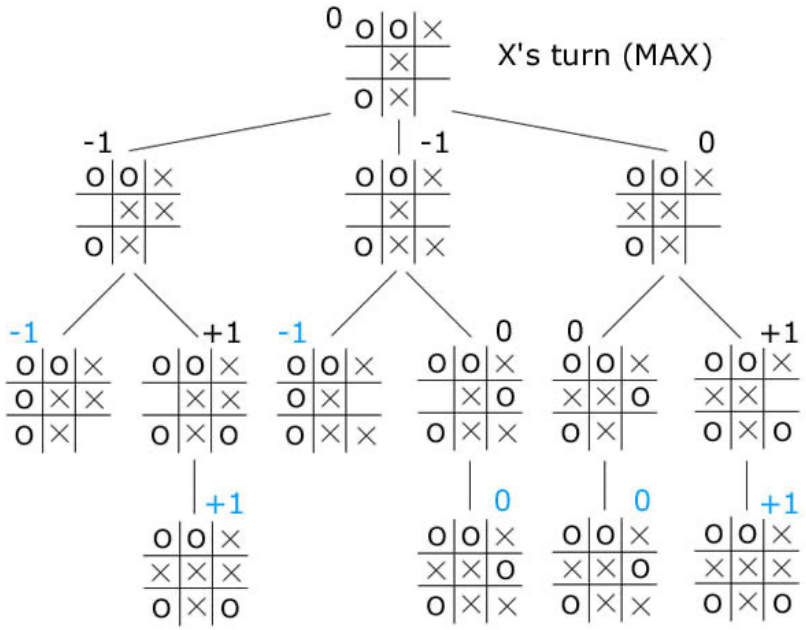
Requirements for the game:

- User chooses to play first or second
- Depending on the choice, the first player makes move with "X"
- Players make moves one by one
- During his/her move, the player chooses the coordinates where to put his mark
- Game finishes if one of the players has 3 of his marks in a row, column, or diagonal
- Game finishes if the board is full, in this case the result of the game is draw

Print board after every move. Write instructions into the console for the user and try to make the game user-friendly. At the end write the result of the game to the console.

You can import libraries and add auxiliary functions if needed.
The code will be tested by different scenarios of the game, so make sure it works properly. **Test the user inputs for correctness**. Make sure that computer strategy is efficient. The grading will also be based on the **percentage of losses of the computer**.

Using the image below as an example, draw 2 other test cases with 3 empty spaces left (so you will have three branches at the beginning). Draw the whole decision tree, and cross-out branches that are filtered out by alpha-beta pruning. Near each of the left variants, write the return value of your evaluation function. Include the images in the report with explanations.



In the report discuss the evaluation function you have used. Will it work well for checkers or chess? Why? If not, why not?

Variant 2

Using the template from the files folder (lab2_v2.py), write a program that plays **"Connect 4"** game with the user using **min-max algorithm with alpha-beta pruning**. The board of game is a **2D matrix** with entries: " " (space) – empty cell, "X" – first player, "O" – second player.

Requirements for the game:

- User chooses to play first or second
- Depending on the choice, the first player makes move with "X"
- Players make moves one by one
- During his/her move, the player chooses the column where drop his mark
- Game finishes if one of the players has 4 of his marks in a row, column, or diagonal
- Game finishes if the board is full, in this case the result of the game is draw

Implement proper evaluation function using functions evaluate_position and evaluate_window given in the code template.

Print board after every move. Write instructions into the console for the user and try to make the game user-friendly. At the end write the result of the game to the console.

You can import libraries and add auxiliary functions if needed. The code will be tested by different scenarios of the game, so make sure it works properly. **Test the inputs of the user for correctness**. Make sure that computer strategy is efficient. The grading will also be based on the **percentage of losses of the computer**.

In the report discuss the evaluation function used for scoring the position of the game. Discuss weaknesses and strengths of the chosen evaluation strategy.

Variant 3

Using the template from the files folder (lab2_v3.py), write a program that plays **"Nim"** game with the user using **min-max algorithm with alpha-beta pruning**. The board of game is **a list** each index of which represents a pile and value represents number of sticks in the pile.

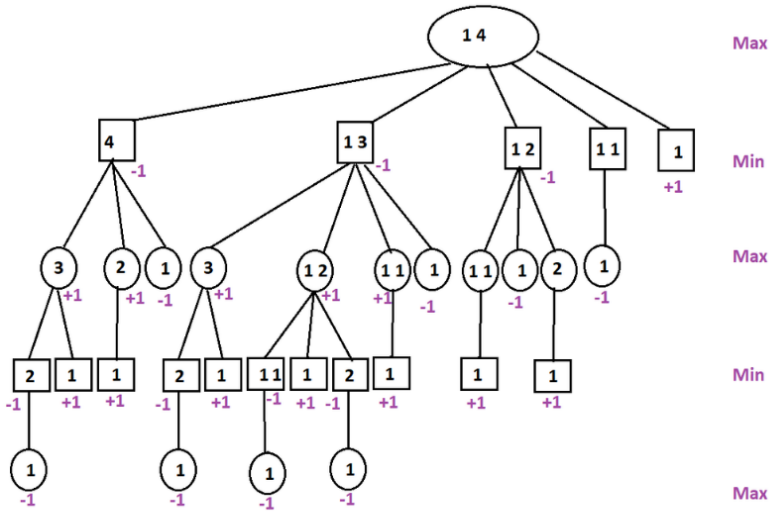
Requirements for the game:

- User enters number of piles and number of sticks in each pile at the beginning of the game
- User starts first
- Players make moves one by one
- During the move, the player chooses the pile and number of sticks to remove from that pile
- The goal is to make the opponent to take the last stick
- Game finishes if 0 or 1 stick is left

Print board after every move. Write instructions into the console for the user and try to make the game user-friendly. At the end write the result of the game to the console.

You can import libraries and add auxiliary functions if needed.
The code will be tested by different scenarios of the game, so make sure it works properly. **Test the inputs of the user for correctness**. Make sure that computer strategy is efficient. The grading will also be based on the **percentage of losses of the computer**.

Using the image below as an example, draw 2 other test cases with 2 piles and at least 4 levels of the decision tree. Draw the whole decision tree, and cross-out branches that are filtered out by alpha-beta pruning. Near each of the left variants, write the return value of your evaluation function. Include the images in the report with explanations.



In the report discuss the evaluation function used for scoring the position of the game. Will it work well for checkers or chess? Why? If not, why not?

Variant 4

Using the template from the files folder (lab2_v4.py), write a program that solves **Sudoku** problem as **constraint satisfaction problem using backtracking with forward checking**. The game board is **9x9 grid** filled with entries 0 (cell is empty), or from 1 to 9. The objective is to return the first found solution to the given sudoku problem.

You can import libraries and add auxiliary functions if needed.
Come up with different test cases and corner cases. The code will be run against different test cases, so make sure it works properly. Test the inputs for the correctness.

Make **step by step visualization** of sudoku solving algorithm.
The interface may be as simple as possible.

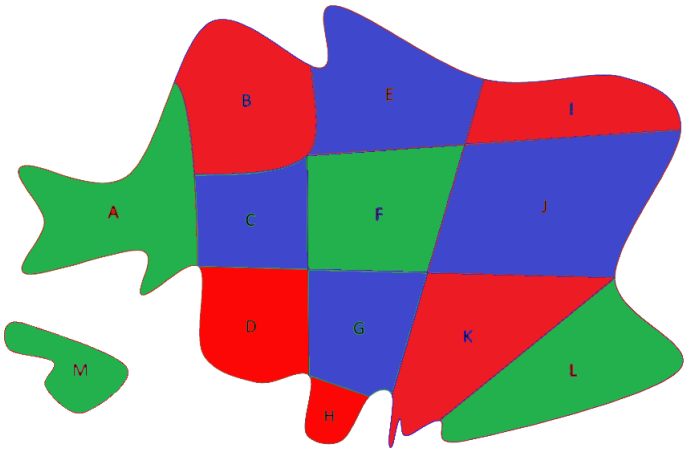
Explain how you defined variables, domains, and constraints. Discuss the role of forward checking, and answer if it improves the algorithm. Add interesting test cases and corner cases in the report and explain the choice of the test case.

Variant 5

Using the template from the files folder (lab2_v5.py), write a program that solves **map coloring** problem as **constraint satisfaction problem using backtracking with forward checking**. The map is a **dictionary** in which each region is mapped to the list of its neighbors. The objective is to return the **first found solution** to the given map coloring problem.

You can import libraries and add auxiliary functions if needed.
Come up with different test cases and corner cases. The code will be run against different test cases, so make sure it works properly. Test the inputs for the correctness.

Similarly to the image below, draw at least two maps with at least 10 regions and color them with found solutions. You can draw and color by hand and take a photo. Alternatively, you can use any application of your choice or write code to generate such images. The result should be included in the report along with the input to the code.



Explain how you defined variables, domains, and constraints. Discuss the role of forward checking, and answer if it improves the algorithm. Add interesting test cases and corner cases in the report and explain the choice of the test case.

2 Technical details

- The submission should include the python file based on the provided template and the report in the format of .pdf
- Please ensure that your code adheres to basic standards of lean coding in accordance with PEP8. Additionally, it should contain comments on the crucial parts to help with **readability and understanding.**
- **Include check for invalid inputs**
- The inputs to the code should be provided inside the python code or from the file. In case of providing inputs from the file, the python code should include reading from the file part and provide instructions in the comments. In addition, the file with the inputs should be included in the submission.
- All the test cases used should be in the submission (v4, v5). The code without proper test cases may get less points.

3 Submission guidelines

You should submit all the files to private GitHub repository not later than:

- 05.04.2025 until **23:59** for Monday group
- 31.03.2025 until **23:59** for Wednesday group
- 02.04.2025 until **23:59**. for Friday group

The on-line assessment will take place during your labs right after the deadline. In case of questions, please contact me via Teams.

4 Assessment criteria

You can get [0, 5] points for the lab. The following criteria will be used to evaluate your work:

- Proper implementation of the algorithm: **1 point**

- Final report including clear explanation of the programmed solutions and discussion of the results: **2 points**
- Explaining the solution and answering questions during the online assessment: **2 points**