UNIVERSITY OF ABERDEEN

University of Aberdeen

School of Natural and Computing Sciences

Department of Computing Science

2024 – 2025

Programming assignment – Groupwork by a team of 4-5 students

Title: JC4004 – Computational Intelligence	Note: This assignment accounts for 30% of the total mark of the course.

Deadline: Submit the assignment in *MyAberdeen* by 19. December 2024 at 23:00 (China time).

Information for Plagiarism and Collusion: The source code and your report may be submitted for plagiarism check in *MyAberdeen*. Please refer to the slides available at *MyAberdeen* for more information about avoiding plagiarism before you start working on the assessment. Excessive use of large language models, such as ChatGPT, for writing the code or the report can also be considered as plagiarism. In addition, submitting similar work with another group can be considered as collusion.

Information about Extensions: According to the new extension policy of University of Aberdeen, teachers are no longer allowed to give deadline extensions for coursework assignments. Extensions may be requested from the school administration by e-mail: uoa-ji-enquiries@abdn.ac.uk.

Extensions require strong justifications (such as serious illness or grievance), and extension requests should be accompanied with supporting evidence, such as a medical certificate. See also a separate document for the extension policy. Since this assignment is a groupwork assignment, extensions would be granted in very exceptional situations only.

Introduction

In this assignment, your task is to build an artificial intelligence game bot for playing the traditional board game ***Fox and Goose***. Your game bot should be able to play the game on both sides, as a fox and as a goose. The detailed rules of the game are explained below. Please note that there are different versions of the game: for this assignment, you should follow the rules described in this document.

Fox and Goose is a two-player board game. One of the players is a fox trying to capture all the geese. Another player represents the geese and tries to surround the fox so that it cannot move any more. The game is played on a board with 33 possible locations for the fox and the geese. In the beginning, there are 15 geese and one fox on the board, as illustrated in the Figure 1. The white pieces are the geese, and the red piece is the fox.

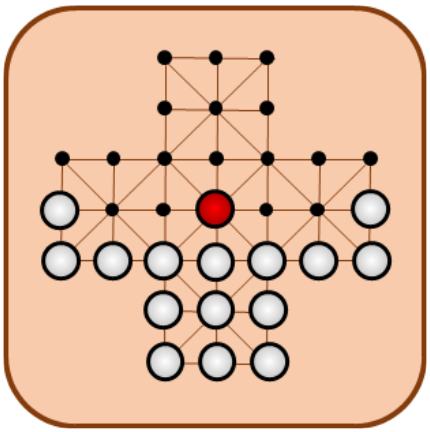

Figure 1. Initial positions in Fox and Goose.

The game is played in turns. In this version of the game, the fox and the geese can both move one step horizontally, vertically, or diagonally on their turn along the lines on the board. The player playing goose can select any of the geese on the board to move. *Please note that diagonal movement is only allowed from some of the positions, as indicated by the lines on the board.*

You cannot move a piece to a position that is already taken by another piece. However, the fox can *capture* (or eat) a goose by jumping over it to a free position. The captured goose is removed from the board. It is also possible to capture multiple geese in one turn by chaining the jumps like in *Checkers*. A goose cannot capture the fox. It is not mandatory to capture even if it is possible, but it is mandatory for both the fox and the geese to make a move in their turn. Examples of legal moves are shown in Figure 2 below.

The goal of the geese is to surround the fox so that it cannot make any legal moves anymore. The goal of the fox is to capture all the geese. Theoretically, the minimum of four geese would be enough to surround the fox; therefore, the fox wins when there are less than four geese left on the board. Examples of winning the game are shown in Figure 3.

Since the fox and the geese have a different goal and follow different rules, the game is *unbalanced*. Therefore, the players usually play an even number of games, swapping their roles. The player that

wins more games is the final winner. In this assignment, your task is to implement the game logic for both the fox and the goose.

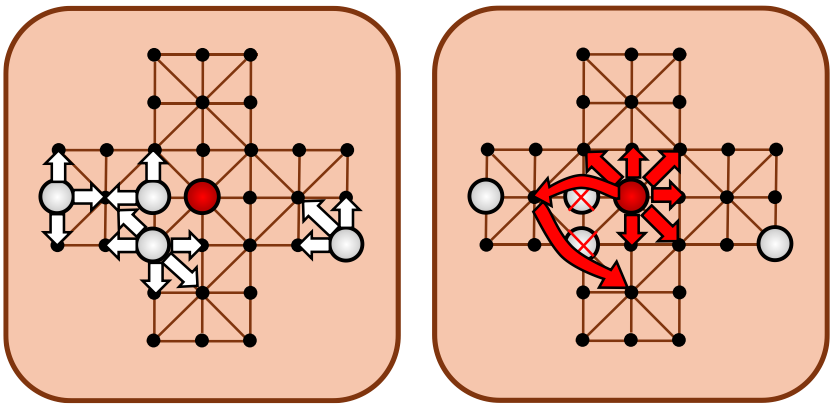

Figure 2. Examples of legal moves for the geese (left) and the fox (right), respectively.

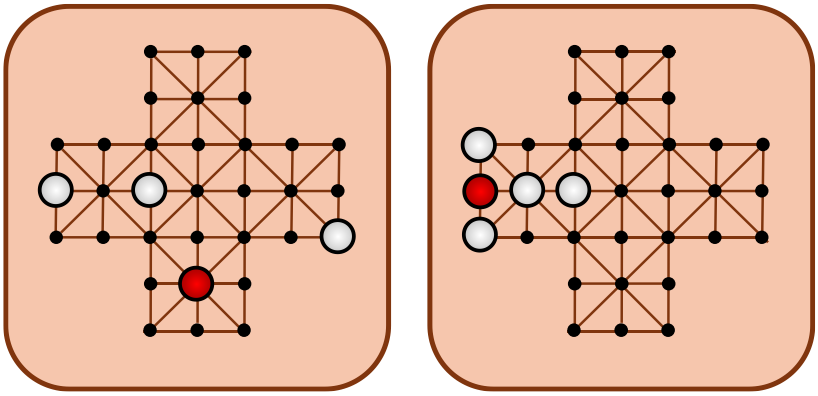

Figure 3. Examples of the fox winning the game (left) and the geese winning the game (right).

General Guidance and Requirements

In this assignment, you are required to write a Python class `Player` that is able to play Fox and Goose game through methods `play_fox()` and `play_goose()`. The current game board is passed to the methods as a parameter, and the methods will return the next move as a fox or as a goose, respectively. Python file **TestFoxAndGoose.py** will be shared to demonstrate how the game testing framework uses the `Player` class.

The board is a 2-D `list` object with 7 × 7 characters representing the state of the game. Characters `'F'` and `'G'` mark the fox and the geese, respectively. An empty position is marked with a dot `'.'` and a space `' '` marks a position that is off the playing area. The board is initialised in class `FoxAndGoose` in file **TestFoxAndGoose.py** as follows:

```
self.board = [
        [' ',' ','.','.','.',' ',' '],
        [' ',' ','.','.','.',' ',' '],
        ['.','.','.','.','.','.','.'],
        ['G','.','.','F','.','.','G'],
        ['G','G','G','G','G','G','G'],
        [' ',' ','G','G','G',' ',' '],
        [' ',' ','G','G','G',' ',' ']
    ]
```

The `play_fox()` and `play_goose()` methods in your code should take the board as defined above as an input parameter. As an output parameter, the method should return a `list` object with two or more pairs of integers, where the first value represents the row, and the second value represents the column on the board. The first pair is the initial position, and the second pair is the target position. For example, return value `[[3,2],[3,3]]` means that the piece in the 4th row, 3rd column will be moved to the 4th row, 4th column. Note that the numbering starts from zero: for example, position `[0,1]` is the 2nd column of the 1st row.

The `play_fox()` method can return a longer list with several target positions in case the fox captures more than just one goose in one move. For example, return value `[[3,3],[3,1],[5,3]]` means that the fox first jumps from position `[3,3]` to position `[3,1]`, capturing the goose in position `[3,2]`, and then continues to position `[5,3]`, capturing the goose in position `[4,2]`.

You can decide freely what kind of techniques of computational intelligence you use to implement the game logic. You can implement additional functions and classes if necessary. However, the `Player` class should interact with the game framework only through the `play_fox()` and `play_goose()` methods, as described above. The bot should have a reasonable complexity: in the testing phase, a time limit of 5 seconds will be applied to consider the moves. If your implementation requires time-consuming initialisation, such as downloading a deep neural network, initialisation should be done in the class `Player` constructor `__init__`, not the `play_fox()` and `play_goose()` methods.

You can use code generation tools and code from external sources moderately for assisting implementation of parts of the code, but the use of any sources or tools should be explained, and the references should be given in the project report.

Submission Requirements

You should submit the work in the course page in *MyAberdeen*. Your submission should include at least two files: file **TeamXX.py** that includes the Python code implementing class `Player` with methods `play_fox()` and `play_goose()`, and **ReportXX.pdf** that is the project report. In the file names, replace **XX** with team number, for example **05**. As an example, we provide file **Team00.py** that allows you to play the game manually with moves entered by a human user. If your code requires any additional files to run, such as pre-trained neural network, you should include them also in your submission.

Please note that it is your responsibility to make sure that the code in **TeamXX.py** works when we test it: you should use file **TestFoxAndGoose.py** to import your class and to test that your code works with the testing framework. Replace module name `Team00` in `module=__import__("Team00")` with your own file name without **.py** extension. If your code has external dependencies requiring additional installations, they should be clearly explained in the project report or **readme** file included in the submission.

Note that the game bots implemented by different groups will play against each other, so it is essential to ensure compatibility. You should use **Python 3**. If you use any third-party packages such as TensorFlow or PyTorch, we recommend using the latest stable version and to avoid using features with known backwards compatibility problems. We suggest starting with a clean environment and to keep track of all the installed packages and their version numbers and reporting them in the project report or **readme** file.

Note that at the time of writing, the latest TensorFlow version is not compatible with the latest stable Python release 3.13.0. Therefore, if you plan to use TensorFlow, the Python version should be **3.12.7** or earlier.

The length of the project report should be approximately 1,500 words. It is recommended to include graphical illustrations, but screenshots of the program code should be avoided. If the code implements some complex algorithms that are difficult to explain otherwise, flowcharts or pseudocode can be used as tools of illustration. The report should include the following sections:

1. Introduction: *about 200 words.*
2. Theoretical basis, including description of the used methods and algorithms with a brief justification why those techniques were chosen: *about 600 words.*
3. Implementation details, including the used libraries and e.g., an UML diagram or a list of the essential methods and their parameters: *about 300 words.*
4. Conclusions, including self-reflection, difficulties faced, experiences from testing the code, and ideas for future improvements: *about 300 words.*
5. Summary of the individual roles, including brief description of team members' contributions: *about 100 words.*
6. References.

If you wish the results for your group to be published in the leaderboard in MyAberdeen, please give a name for your group in the report!

Marking Criteria

The assignment will be marked based on the ***project report*** (40 marks), ***methodology*** (40 marks), and ***performance*** (20 marks).

The project report will be marked according to the coverage of the required aspects, clarity of presentation (including language and illustrations), consistency between the report and the submitted code, and relevance of the references.

The methodology will be evaluated based on the suitability of the chosen methods and algorithms for the given task, creativity (for example, combining different methods in an unconventional way), and implementation (e.g., clarity of the source code, computational efficiency).

For performance evaluation, we will test all the submitted assignments by arranging them to play against each other. Every submission will play against each of the other submissions twice, once as a fox and once as a goose. The results will be aggregated in a league table, where a win gives one point, and a loss gives zero points. The winner will be awarded 20 marks, and the other groups will be awarded marks based on the formula:

$$m_i = \frac{20x_i}{x_{winner}},$$

where m_i is the mark for group i, x_i is the total points for group i, and x_{winner} is the total points for the winner of the league.

Note that if both game bots repeat moves back and forth to the same position, the game may end in a deadlock situation. To resolve deadlocks, the maximum number of moves is set to 1000. If the game ends without a winner due to a deadlock, both players will be awarded zero points.

Contact

For any questions or clarifications, you can contact the course teachers: Dr Jari Korhonen (jari.korhonen@abdn.ac.uk) for AI, and Dr Yongchao Huang (yuan.wen@abdn.ac.uk) for CS.