UNIVERSITY OF ABERDEEN	**University of Aberdeen** **School of Natural and Computing Sciences** **Department of Computing Science** **2025 – 2026**

Programming assignment – Groupwork by a team of 3-5 students

Course: 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 2025 at 22:00 (China time).

Information for Plagiarism, Collusion, and use of Generative AI: 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 and the responsible use of generative AI (GenAI) tools before you start working on the assessment. Excessive use of GenAI tools, like ChatGPT or DeepSeek, for writing the code or the report, can 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 ***Asalto*** (*Assault*). Description of the game can be found on the internet, e.g., http://www.cyningstan.com/game/275/asalto. 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.

Asalto is a two-player asymmetric board game. One of the players represents ***rebels*** trying to occupy the fortress, and the other player represents ***officers*** trying to defend the fortress. The game is played on a board with 33 possible locations for the rebels and officers. In the beginning, there are 24 rebels and two officers on the board, as illustrated in the Figure 1. The blue pieces are the rebels, and the white pieces are the officers. The nine positions shown with darker background in the Figure 1 is the fortress.

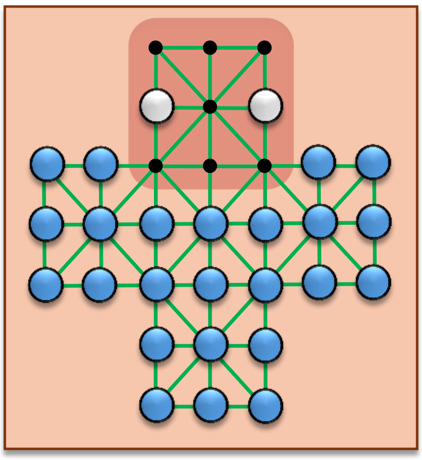
Figure 1. Initial positions in Asalto.

The game is played in turns. Officers can move one step horizontally, vertically, or diagonally on their turn along the green lines drawn on the board. *Please note that diagonal movement is only allowed along the lines shown on the board.* Rebels move in the similar way, but *they cannot move away from the fortress*. Therefore, the rebels cannot move downwards, and they can only move sideways and diagonally towards the middle column, unless they are in the middle column already. The rebels cannot move downwards even inside the fortress.

Neither rebels nor officers cannot move to a position that is already taken by another piece. However, the officers can *capture* a rebel by jumping over it vertically, horizontally, or diagonally, to a free position. The captured rebel is removed from the board. It is also possible to capture multiple rebels in one turn by chaining the jumps like in *Checkers*. A rebel cannot capture an officer. Note that *it is mandatory to capture* if it is possible: if an officer fails to capture a rebel, the officer will be removed from the board (huffed). One capture is enough to avoid huffing, even if the officer could capture more than one rebel. Examples of legal moves are shown in Figure 2 below.

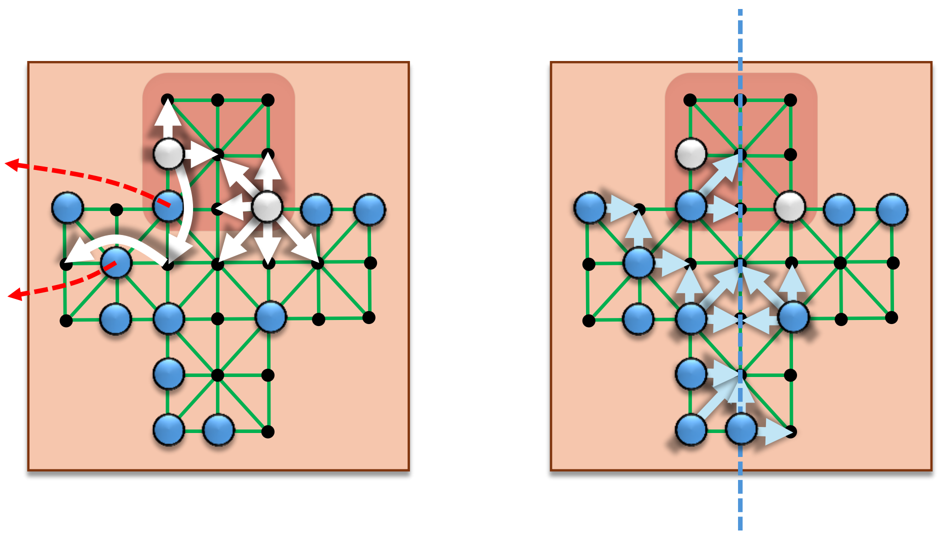

Figure 2. Examples of legal moves for the officers (left) and the rebels (right), respectively.

The goal of the rebels is to occupy all the nine positions in the fortress. The rebels will also win the game if they surround the officers so that they cannot make any legal moves anymore, or if both officers are huffed and there are no officers left. The goal of the officers is to capture so many rebels that they cannot win the game anymore. Theoretically, the minimum of nine rebels would be enough to occupy the fortress; therefore, the officers win when there are less than nine rebels left on the board. Examples of winning the game are shown in Figure 3.

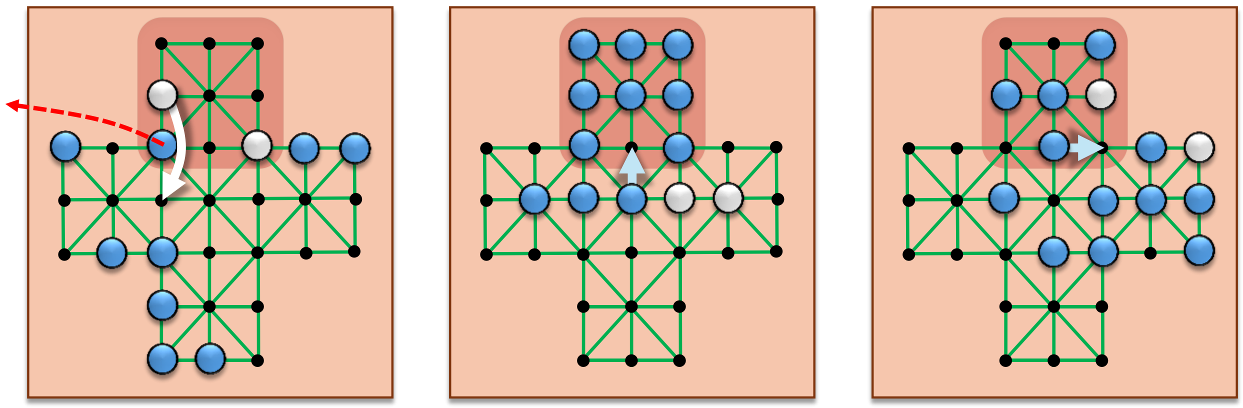

Figure 3. Examples of the ending the game. Left: officer captures a rebel, and then there are only eight rebels left, so the officers win. Middle: after the move, rebels occupy all the nine positions in the fortress and win the game. Right: after the move, rebels block officers from any legal moves, so the rebels win the game.

Since rebels and officers have a different goal and follow different rules, the game is *unbalanced*. Therefore, the players will play an even number of games, swapping their roles each round. The player that wins more games is the final winner. In this assignment, your task is to implement the game logic for both rebels and officers.

General Guidance and Requirements

In this assignment, you are required to write a Python class `Player` that can play Asalto game through methods `play_rebel()` and `play_officer()`. The current game board is passed to the methods as a parameter, and the methods will return the next move as a rebel or as an officer, respectively. Python file **Asalto.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 `'R'` and `'O'` mark the rebels and the officers, 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 `Asalto` in file **Asalto.py** as follows:

```
self.board = [
        [' ',' ','.','.','.',' ',' '],
        [' ',' ','O','.','O',' ',' '],
        ['R','R','.','.','.','R','R'],
        ['R','R','R','R','R','R','R'],
        ['R','R','R','R','R','R','R'],
        [' ',' ','R','R','R',' ',' '],
        [' ',' ','R','R','R',' ',' ']
    ]
```

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

Some specific rules should be observed when building the bot:

- If an officer fails to capture a rebel when possible, the officer is automatically huffed, i.e., removed from the board.
- If the bot makes an illegal move, the turn will be lost, but the game continues normally.
- To avoid deadlocks, the maximum number of rounds is 1000. If the game is not finished after 1000 rounds, the winner is the player with less illegal moves. If the number of illegal moves is the same, the result is a tie, and both players receive zero points.

- If `play_rebel()` or `play_officer()` method throws an exception, the bot will automatically lose the game.
- Time limit will be applied. If the bot uses more than 10 seconds for a move, it will automatically lose the game.

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_rebel()` and `play_officer()` methods, as described above. 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 in the `play_rebel()` and `play_officer()` 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_rebel()` and `play_officer()`, and **ReportXX.pdf**, that is the project report. In the file names, replace **XX** with team number, for example **05**. If your code requires any additional files to run, 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 **Asalto.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.

The game bots will play a tournament against each other. Therefore, ***it is required that all the implementations use compatible Python libraries.*** You can find file **requirements.txt** in *MyAberdeen* to configure your Python environment. The commonly used standard libraries for machine learning, such as ***numpy***, ***pytorch***, and ***tensorflow***, are included. To ensure compatibility, ***the bot MUST NOT use any additional external libraries!*** Game bots that cannot be tested using the default configuration will receive zero marks for performance. You can use additional libraries for training the model, but your code in **TeamXX.py** must run without any other libraries than those defined in **requirements.txt**. Your code should work directly when the files in your submission are extracted in the same folder with **Asalto.py** file.

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 and acknowledgements. If you have used any GenAI tools in the work, please mention them in the acknowledgements.

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 rebels and once as officers. 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 1,000. If the game ends without a winner due to a deadlock, winner will be decided from the number of illegal moves. If both teams have the same number of illegal moves, both teams will receive zero points.

Contact

For any questions or clarifications, you can contact the course teachers: Dr Yongchao Huang (yongchao.huang@abdn.ac.uk), Dr Shahzad Mumtaz (shahzad.mumtaz@abdn.ac.uk), or Dr Jari Korhonen (jari.korhonen@abdn.ac.uk).