FIT9136 Algorithms and Programming Foundations in Python

Assignment 1

Last Updated: 23 July 2023

Table of Contents

1. Key Information

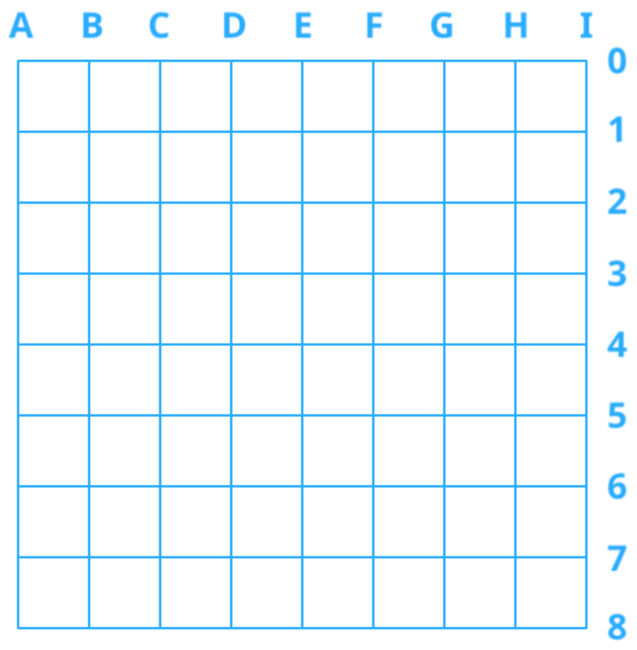
Purpose	This assignment will develop your skills in designing, constructing, and documenting a small Python program according to specific programming standards. This assessment is related to (part of) the following learning outcome (LO): ● **LO1:** Apply best practice Python programming constructs for solving computational problems
Your task	This assignment is an Individual task where you will write Python code for a simple application whereby you will be developing a simple board game as per the specification.
Value	**25%** of your total marks for the unit.
Due Date	**Friday, 25 August 2023, 4:30 PM (AEST)**
Submission	● Via Moodle Assignment Submission. ● FIT GitLab check-ins will be used to assess the history of development ● Turnitin will be used for similarity checking of all submissions.
Assessment Criteria	This assessment includes a compulsory interview with your tutor following the submission date. At the interview you will be asked to explain your code/design/testing, modify your code, and discuss your design decisions and alternatives. Marks will not be awarded for any section of code/design/functionality that you cannot explain satisfactorily. Failure to attend the interview will result in your assessment not being marked. You will be provided with the timing of the interviews at a later date. The following aspects will be assessed: 1. Program functionality in accordance to the requirements 2. Code Architecture and Adherence to Python coding standards 3. The comprehensiveness of documented code and test strategy
Late Penalties	● 10% deduction per calendar day or part thereof for up to one week ● Submissions more than 7 calendar days after the due date will receive a mark of zero (0) and no assessment feedback will be provided.
Support Resources	See Moodle Assessment page and Section 8 in this document
Feedback	Feedback will be provided on student work via ● general cohort performance ● specific student feedback ten working days post submission

2. Context Information

For this assignment, you will be required to create a Python program that simulates the board game *Gomoku*. *Gomoku*, also referred to as *Five in a Row*, is a popular board game originating from East Asia. Two players strategically play on a square grid. The objective for both players is to be the first one to create a continuous line of five stones either horizontally, vertically or diagonally. The players alternate turns to place stones on vacant intersections of the grid, where they need to create their line of five stones while stopping the opponent from achieving the same.

2.1. The Game Board

This game involves a rectangle board with different board settings. An example 9 * 9 board is shown below :



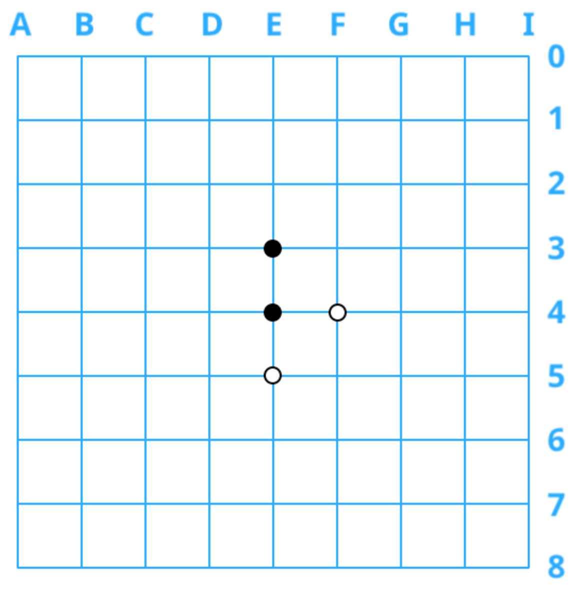
We will use the same coordinate system as indicated in the above example:

1. Rows increase from the top to the bottom and numerical indices are used (e.g., for a 9 * 9 board, the top row is row 0 and the bottom row is row 8);
2. Columns increase from left to right and uppercase letters are used as indices(e.g., for a 9 * 9 board, the leftmost column is column A and the rightmost column is column I).

Please note that, stones are placed at the intersections (e.g., 0A, 0B, 1A, 1B).

2.2. Taking Turns

Players take turns dropping a coloured stone (i.e., black/white) onto one of the intersections as shown below:



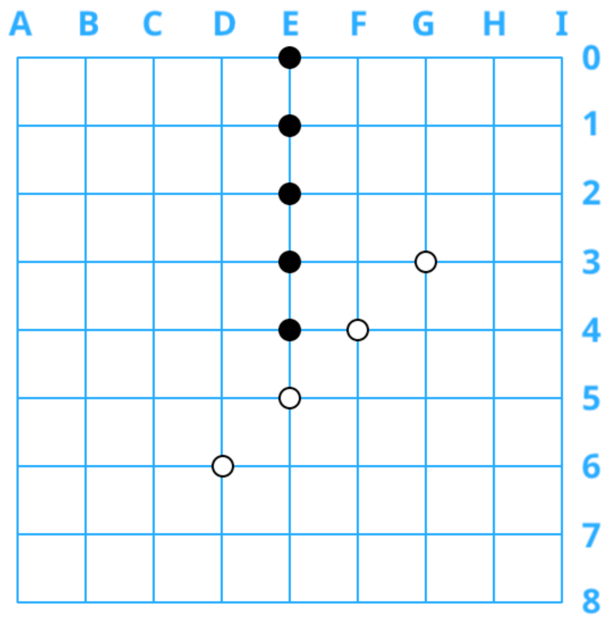
By convention, the player using the **black** stones begins the game by placing one of their pieces on the board. Here, the same convention is followed.

2.3. Ending a Game

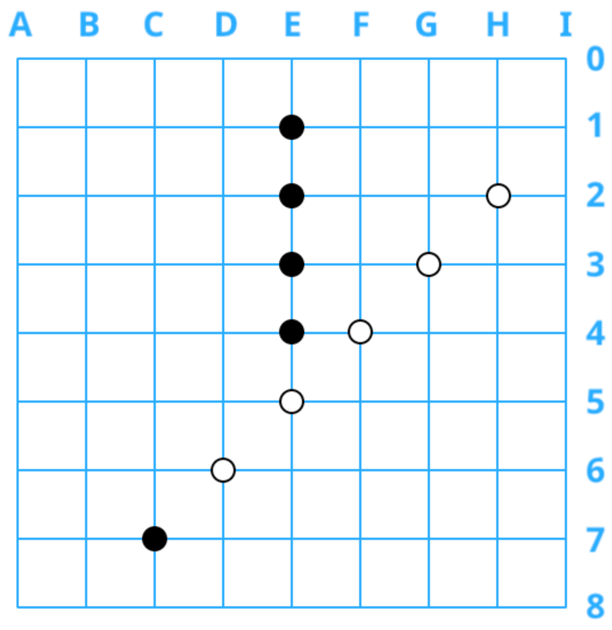
Players take turns to place stones on the board until either

- One player has an unbroken line of five stones of their colour. The line can be either horizontal, vertical or diagonal.
- The board is completely filled (i.e. all spots have been occupied) and both players are unable to make a continuous line of five stones. In this case, the game ends in a draw.

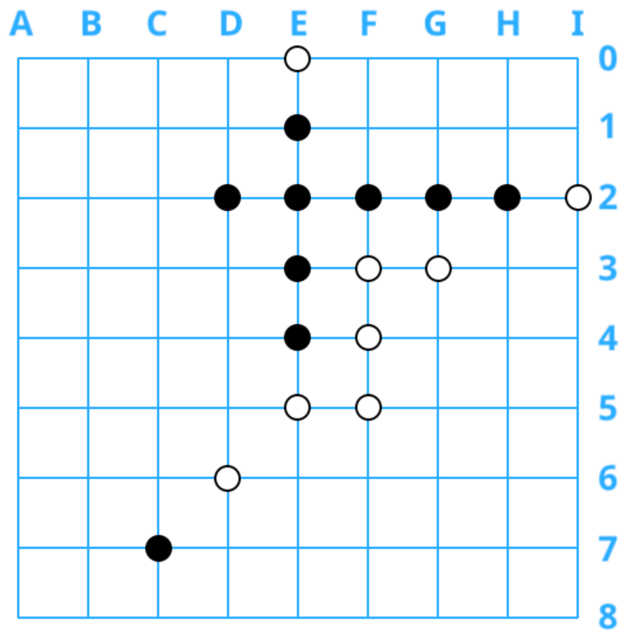
Some examples of winning states:



Player 1 (black) has a vertical win in column E, row 0 - row 4 (i.e., all stones in these spots are black)



Player 2 (white) has a diagonal win in row 2 column H, row 3 column G, row 4 column F, row 5 column E and row 6 column D (i.e., all stones in these spots are white)



Player 1 (black) has a horizontal win in row 2, columns D-H (i.e., all stones in these spots are black)

2.4. Your Task

You will eventually construct a program that simulates the players playing this game but it will be broken into small stages to help you develop your program.

2.5. Video Description

This video gives a good description of the game: Link to video.

3. Implementation Instructions

Your implementation **must include a text interface with all necessary functionalities** as detailed below. Your program will be evaluated based on its ease of use and clarity, including the provision of clear information and error messages for the player.

The objective of your implemented *Gomoku* game is to allow 1) a player to play against a simple computer player; and 2) a player to play against another player. Victory is attained by the first player who achieves an unbroken line of five stones of their colour. The task at hand requires the creation of functions to facilitate the entire gameplay process. It is imperative to carefully review the following comprehensive regulations and prerequisites for each function and aim to execute them.

To ensure that your code can be properly evaluated by the teaching team:

1. **Please ensure that your implemented functions use the same names and function signatures (i.e. number and type of input arguments and the type of the return value) as required,**
2. **Please ensure that your code is properly formatted, such as proper variable naming conventions, consistent indentations, proper line length, etc.,**
3. **Please ensure that you provide clear and coherent comments on your code to aid with the graders' interpretation of your code,**
4. **For more details regarding formatting and commenting, see the PEP 8 Style Guideline.**

3.1. Game menu function

The `game_menu()` function is responsible for displaying the game menu at the start of the game, as well as during the game process to provide instructional suggestions. The menu should at least include the following five options:

1. Start a Game
2. Print the Board
3. Place a Stone
4. Reset the Game
5. Exit

This function accepts zero parameters and has no return value.

3.2. Creating the Board

The purpose of this function is to create a data structure which will be used to keep track of the states of the boards. Your task here is to write the `create_board(size)` function, where `size` is the size of the board (e.g. to create a 9 by 9 board, we call `create_board(9)`). You should carefully decide the data structure to be used as well as the values to be stored to represent the unoccupied intersections of the grid.

This function returns the data structure which will be used to keep track of the occupancy status of the intersections on the board. The occupancy status of the intersections for the newly created board should all be unoccupied.

3.3. Is the target position occupied?

The purpose of this function is to examine whether a specific position on the board is occupied by a stone. Your task is to write a function `is_occupied(board, x, y)` where `board` is the current state of the board, `x` is the row index and `y` is the column index. Here, you can assume that `x` and `y` are both valid numeric indices (i.e., both `x` and `y` are greater than or equal to 0 and smaller than the size of the board).
This function returns a boolean value of either `True` or `False`.

3.4. Placing a Stone at a Specific Intersection

Now we want to replicate placing a stone on the board. Your task is to write a function `place_on_board(board, stone, position)` which:

- Takes the following parameters:
  - A board
  - A stone value (either "●" or "○")
  - A position (a tuple of **strings** (a, b) where a is the row index, e.g., "0", and b is the column index, e.g., "A")
- If the move is successfully performed, return a boolean value of `True`
- If the move is impossible (e.g., invalid or occupied position), return a boolean value of `False`

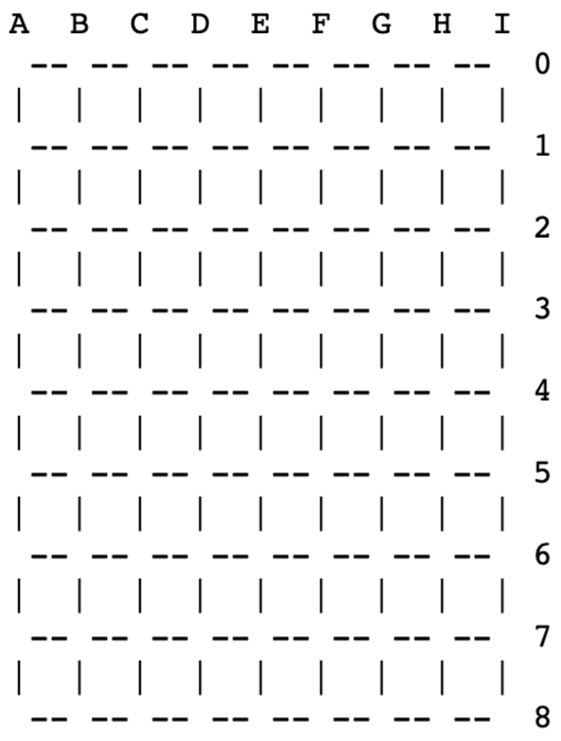
You will need to invoke functions implemented in previous steps.

3.5. Printing the Board

In order for players to know the states of the current boards, there needs to be a way to visualise the board. Your task here is to write a function `print_board(board)`:

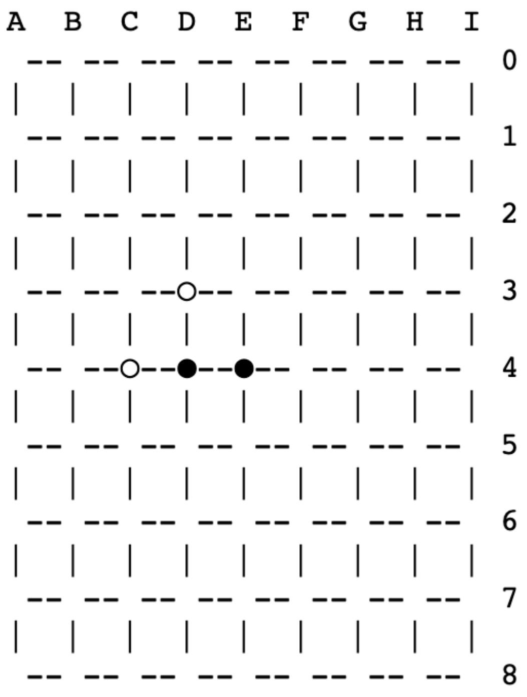
- The parameter `board` can be the ones created and manipulated in previous steps
- "`--`" and "`|`" should be used to represent the grid of the board.
- Apart from the board, the indices of the board need to be presented to enhance user friendliness as indicated in the examples below
- This function does not return any value.

Examples:



```
A  B  C  D  E  F  G  H  I
 -- -- -- -- -- -- -- --  0
|  |  |  |  |  |  |  |  |
 -- -- -- -- -- -- -- --  1
|  |  |  |  |  |  |  |  |
 -- -- -- -- -- -- -- --  2
|  |  |  |  |  |  |  |  |
 -- -- -- -- -- -- -- --  3
|  |  |  |  |  |  |  |  |
 -- -- -- -- -- -- -- --  4
|  |  |  |  |  |  |  |  |
 -- -- -- -- -- -- -- --  5
|  |  |  |  |  |  |  |  |
 -- -- -- -- -- -- -- --  6
|  |  |  |  |  |  |  |  |
 -- -- -- -- -- -- -- --  7
|  |  |  |  |  |  |  |  |
 -- -- -- -- -- -- -- --  8
```

An empty 9 * 9 board printed by this function



```
A  B  C  D  E  F  G  H  I
 -- -- -- -- -- -- -- --  0
|  |  |  |  |  |  |  |  |
 -- -- -- -- -- -- -- --  1
|  |  |  |  |  |  |  |  |
 -- -- -- -- -- -- -- --  2
|  |  |  |  |  |  |  |  |
 -- -- --○-- -- -- -- --  3
|  |  |  |  |  |  |  |  |
 -- --○--●--●-- -- -- --  4
|  |  |  |  |  |  |  |  |
 -- -- -- -- -- -- -- --  5
|  |  |  |  |  |  |  |  |
 -- -- -- -- -- -- -- --  6
|  |  |  |  |  |  |  |  |
 -- -- -- -- -- -- -- --  7
|  |  |  |  |  |  |  |  |
 -- -- -- -- -- -- -- --  8
```

A 9 * 9 board with both players placing two stones printed by this function

3.6. Check Available Moves

The purpose of this function is to find out whether there are available moves on the board. Your task is to write a function `check_available_moves(board)` where `board` is the current state of the board. This function returns the available moves as a list of tuples where each tuple represents a position on the board, e.g., `[("0", "A"), ("3", "D")]`. If the board is full, return an empty list.
You will need to invoke functions implemented in previous steps.

3.7. Check for the Winner

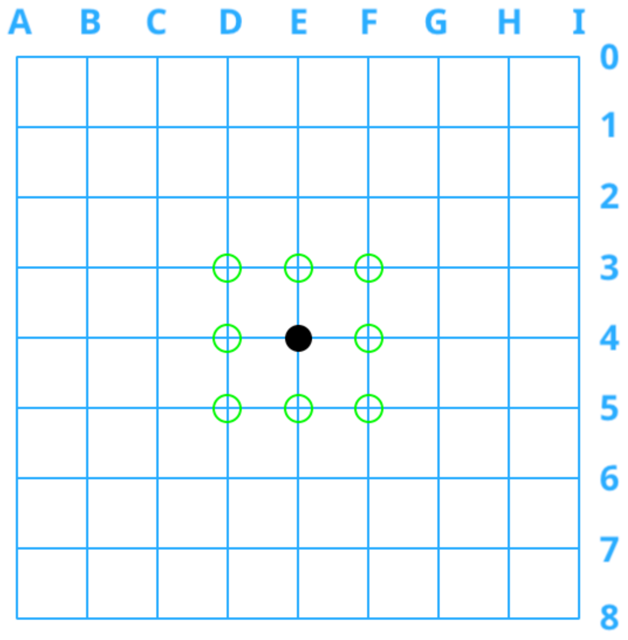
The purpose of this function is to identify the winner of the game. A winning condition is achieved when one of the players successfully forms a continuous line of five stones in their colour, either horizontally, vertically or diagonally. Your task is to write a function `check_for_winner(board)`:

- The parameter `board` can be the ones created and manipulated in previous steps
- If a continuous line of five stones in the same colour has been achieved, return the corresponding stone
- If the board is full but none of the players achieves a continuous line of five stones, return a string value of "`Draw`"
- If none of the players achieve the winning condition and there are still available moves on the board, return `None`

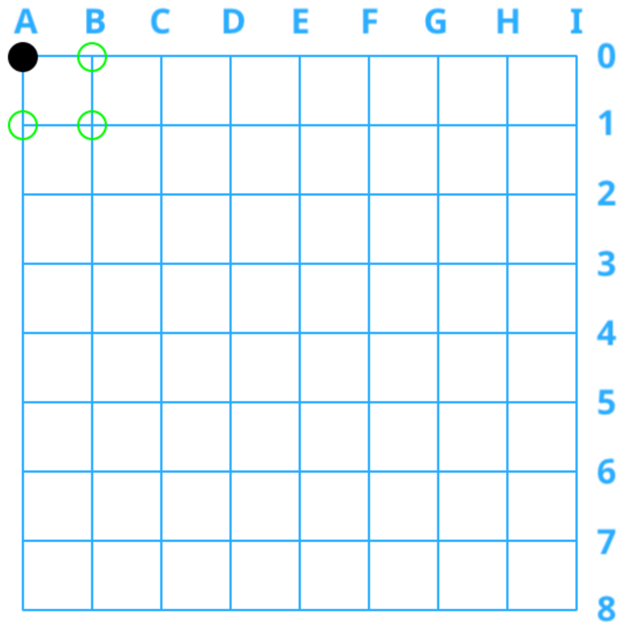
You will need to invoke functions implemented in previous steps.

3.8. Random Computer Player

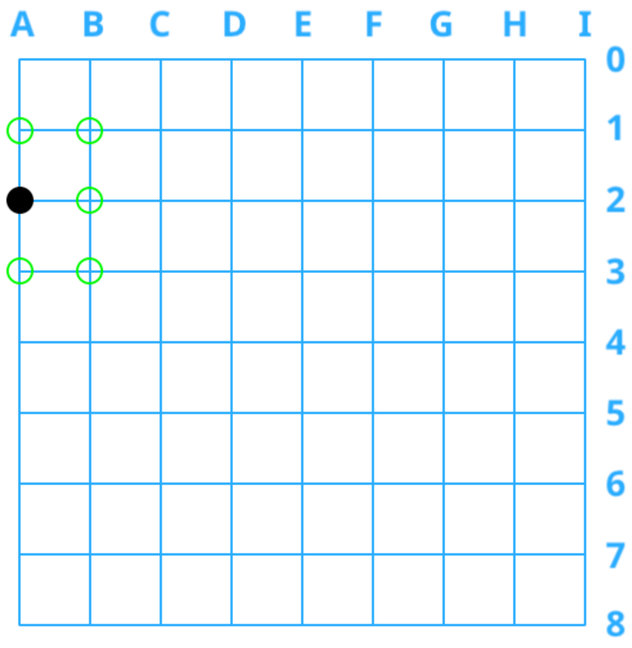
The purpose of this function is to implement a computer opponent. The computer opponent will counter the player by randomly selecting one of the available moves around the player's previous move. For example:



When the player places the stone at (“4”, “E”), the green spots surrounding the player stone suggest the moves the computer player can choose from. **<u>From these green spots, the computer player can only choose one of the spots that are unoccupied.</u>**



When the player places the stone at ("0", "A"), the green spots surrounding the player stone suggest the available moves for the computer player to choose from. Note that, same as previously mentioned, the computer player can only choose one of the green spots that are unoccupied.



When the player places the stone at ("2", "A"), the computer player can only choose one of unoccupied spots from the green spots surrounding the player stone as shown in the figure.

The available moves for the computer player are based on both the player's previous move and the availability of that move's surrounding positions. Your task here is to implement the function `random_computer_player(board, player_move)`:

- The `player_move` is in the same position format, e.g., `("0", "B")`
- Seeing the position of the player's previous move as the centroid, the function needs to find all the available **valid** positions within a 3 * 3 square and randomly select one of these positions
- If all positions within the 3 * 3 square are invalid, the function should randomly select from one of the available positions on the board
- When randomly selecting the moves, you need to ensure all identified positions have the same likelihood of being selected
- This function should return a tuple of **strings** which represents the next played position for the computer player, e.g., `("1", "D")`

You will need to invoke functions implemented in previous steps.

3.9. Play Game

The purpose of this function is to manage all aspects of the game play. You should invoke all functions implemented in previous steps here. Your task is to implement the function `play_game()`:

- This function accepts zero parameters.
- When the function is invoked, it should display a menu to display all possible options for the user
- Once the user selects an option, the program should execute the relevant function(s) or display additional prompts/ask for additional inputs as needed
- **The user should be able to return to the main menu at any time**
- When the user input option “1”, the function should:
    - Ask for a board size value from the user, the program should at least support size values of 9, 13 and 15.
    - Then, ask for a mode from the user, the mode should be either **Player vs. Player** or **Player vs. Computer**
    - Create a board based on the user specified size
    - If the user input option “1” while a game is in progress, print the instructional message to ask the user either 1) to reset and restart a game, or 2) to complete the current game
- When the user input option “2”, the function should visualise the current state of the board to the user
- When the user input option “3”, the function should:
    - Ask the user to place a stone at a position, the expected input format for the position should be
      “`[row_index] [column_index]`”
      (e.g., “`2 F`”)
    - The user should open the game with the black stone (i.e., "●"). You may need to keep track of the turns to determine the colour of stones to be placed next.
    - Whenever the user successfully places a stone on the board, you should check if any of the players has achieved one of the winning conditions
    - For **Player vs. Player** mode, only one stone from the corresponding player will be placed on the board
    - For **Player vs. Computer** mode, after playing and checking the player’s move, the computer player should play the move as described in section 3.8 and conduct similar checking of the winning conditions
    - If an ending condition is achieved (i.e., either one of the players win, or no more move is available), the game needs to automatically print the result (either print the stone value of the winner or inform the player of a draw game). Subsequently, both the board and the mode need to be reset

- When the user input option “4”, the function should reset the game (i.e., reset the board and the selected mode)
- When the user input option “5”, the function should exit the program. **This is the only situation when the program terminates. The program should continue to run infinitely unless the user selects this option.**

For this function, you will need to validate the inputs from the users and provide meaningful instructional messages. You will need to ensure a clear logic for the implemented function.

4. Do and Do NOT

Do	Do NOT
● Maintain appropriate citing and referencing[1], ● Get support early from this unit and other services within the university, ● Apply for special consideration or for extensions[2] early if needed.	● Leave your assignment in draft mode (assignments in draft mode will not be marked), ● Submit late (10% daily penalty applies)[3], ● Attempt to submit after 7 days of the due date (they will not be accepted), unless you have special consideration.

4.1. Important Notes:

- If **any exception/errors** happen when running each function, you will lose 50% of allocated function logic marks. For example, if the total mark of one function is 10 marks and any exception happens when running this function, then the maximum mark you can get is 5 instead of 10 in the function logic.

- Add correct validation and output messages to make your code more user-friendly to users.

- For each function, add code to test each function and also the expected output. The test code for each function should be placed right after each function and commented (Please refer to the template). For example, the code to test the `check_available_moves` function can be like the following:

```
# b = create_board(9)
# len(check_available_moves(b))
# this should output 81
```

  Note: There is no need to provide the testing for the play_game() function.

- This is an individual assignment and must be completed on your own. You must attribute the source of any part of your code that you have not written yourself. Please note the section on Academic Integrity in this document.

- The assignment must be done using the **Jupyter Notebook**, **Python Version 3.10**.

- The Python code for this assignment must be implemented according to the PEP 8-Style Guide for Python Code.

[1]https://www.monash.edu/library/help/citing-and-referencing/citing-and-referencing-tutorial
[2] https://www.monash.edu/exams/changes/special-consideration
[3] e.g.: If the original mark was 70/100, submitting 2 days late results in 50/100 (20 mark deduction). This includes weekends and public holidays.

- The only allowed library is **random**. You will receive penalties if you use any other libraries.
- For using any built-in function, students should seek confirmation from the teaching team.
- Commenting on your code is an essential part of the assessment criteria. In addition to inline and function commenting on your code, you should include comments/markdowns at the beginning of your program file which specify your full name, student ID, the creation date, and the last modified date of the program, as well as a high-level description of the program.
- This assignment cannot be completed in a few days and requires students to apply what we learn each week as we move closer to the submission date.
- You must keep up to date with the Moodle Ed Assignment 1 forum where further clarifications may be posted (this forum is to be treated as your client). If there are any changes to the specification, apart from making announcements on Ed, we will update the new specification on Moodle.
- Please be careful to ensure you do not publicly post anything which includes your reasoning, logic or any part of your work to this forum, as doing so violates Monash plagiarism/collusion rules and has significant academic penalties. Use private posts or email your allocated tutor to raise questions that may reveal part of your reasoning or solution.
- **In this Assessment, you must NOT use generative artificial intelligence (AI) to generate any materials or content in relation to the assessment task.**

5. Submission Requirements

The assignment must be submitted by **Friday, 25 August 2023, 4:30 PM (AEST)**.

Please ensure that you use the provided template file **(i.e., RENAME-ME.ipynb)** and start working on your assignment as soon as you can.

The following files are to be submitted on Moodle:

- A Jupyter Notebook file (i.e., .ipynb file) that you created to implement your assignment (i.e, code and documentation). Name the file **ass1_studentID.ipynb**
- A PDF file. Use Jupyter Notebook to export a PDF file (Read the instruction provided on Week 2 Applied Class Activities, section 2.2. "**How to Export a Jupyter Notebook to a PDF file?**". Note, The pdf file cannot be an image file. Make sure all the text in the pdf file can be selected and copied). Name the file **ass1_studentID.pdf**

Do not zip these files into one zip archive, submit two independent files. The **.ipynb** file must also have been pushed to the FITGitLab server with an appropriate history as you developed your solutions (a minimum of four pushes, however, we would strongly recommend more than this). Please ensure your committed comments are meaningful.

- No submissions will be accepted via email,
- Please note we **cannot mark any work on the GitLab Server**, therefore you need to ensure that you submit correctly via Moodle since it is only in this process that you complete the required student declaration without which work cannot be assessed.
- It is your responsibility to **ENSURE** that the submitted files are the correct files. We strongly recommend after uploading a submission, and prior to actually submitting in Moodle, that you download the submission and double-check its contents.
- Please **carefully** read the documentation under the "**Special Consideration**" and "**Assignment Task Submission**" on the Moodle Assessments page which covers things such as extensions, correct submission, and resubmission.
- Please note, if you need to resubmit, you cannot depend on your tutors' availability, for this reason, please be VERY CAREFUL with your submission. **It is strongly recommended that you submit several hours before due to avoid such issues.**
- **There are no restrictions on creating extra functions. Do NOT create redundant functions.**
- **In case you have performed any optimizations in your code, please ensure to include a list of these optimizations along with a brief explanation for each in the Markdown cell provided at the bottom of your Jupyter Notebook file (i.e., "Documentation of Optimizations" section in RENAME-ME.ipynb file).**
- **Marks will be deducted for any of these requirements that are not strictly complied with.**

6. Academic Integrity

Students are expected to be familiar with the University Academic Integrity Policy and are particularly reminded of the following:

Section 1.9:

Students are responsible for their own good academic practice and must:

- undertake their studies and research responsibly and with honesty and integrity;
- credit the work of others and seek permission to use that work where required;
- not plagiarise, cheat or falsify their work;
- ensure that their work is not falsified;
- not resubmit any assessment they have previously submitted, without the permission of the chief examiner; appropriately acknowledge the work of others;
- take reasonable steps to ensure that other students are unable to copy or misuse their work; and
- be aware of and comply with University regulations, policies and procedures relating to academic integrity.

and **Section 2.9:**

Unauthorised distribution of course-related materials: Students are not permitted to share, sell or pass on to another person or entity external to Monash:

2.9.1 any course material produced by Monash University (such as lecture slides, lecture recordings, class handouts, assessment requirements, examination questions; excluding Handbook entries) as this is a breach of the Copyright Compliance Policy and such conduct may be a copyright law infringement subject to legal action; or

2.9.2 any course-related material produced by students themselves or other students (such as class notes, past assignments), nor to receive such material, without the permission of the chief examiner. The penalties for breaches of academic misconduct include

- a zero mark for the assessment task
- a zero mark for the unit
- suspension from the course
- exclusion from the University.

Where a penalty or disciplinary action is applied, the outcome is recorded and kept for seven years, or for 15 years if the penalty was excluded.

7. Marking Guide

Your work will be marked as per the following:

- **Functions Implementation - 70 Marks**
    - game_menu() - 1 Mark
    - create_board(shape) - 5 Marks
    - is_occupied(board, x, y) - 4 Marks
    - place_on_board(board, stone, position) - 9 Marks
    - print_board(board) - 6 Marks
    - check_available_moves(board) - 6 Marks
    - check_for_winner(board) - 9 Marks
    - random_computer_player(board, player_move) - 10 Marks
    - play_game() - 20 Marks
- **Optimised Python program - 10 Marks (This section will be assessed only if you score full marks in the Function Implementation section.)**
    - The game is easy to follow with clear information/error messages to the player.
    - The code is efficiently crafted (e.g., using a loop instead of a series of if-else statements, using appropriate data types)
    - Have some creativity in the functionality design, for example:
        - Using GUI,
        - Adding brilliant creative idea on menu (e.g., suggest best move to increase the chance of winning)
- **Interview (compulsory) - 20 Marks**
    - Missing Interview: If you miss the interview, you will receive zero marks for Assignment 1.
    - Interview Score of 5 Marks: If you score 5 marks in the interview, 50% of the marks assigned to the function implementation will be deducted. For example, if you initially scored 70 marks out of 70 for the Functions Implementation and your interview mark is 5, you will only receive 35 marks for the Functions Implementation.
    - Interview Scores of 10, 15, or 20 Marks: If you score 10, 15, or 20 marks in the interview, this mark will be considered as your interview score and added to your final mark.
- **Penalty - up to 20 marks**[4]
    - Missing requirements listed in Section 4. Do and DO NOT and section 5. Submission Requirements.

FinalAssignmentMarkCalculation:

- 80 marks from the Assignment1 submission => 20/25 **PLUS**
- 20 marks from the interview => 5/25

[4] The penalty here is excluded from the late submission

Total: 100 marks, recorded as a grade out of 25

8. Getting help

8.1. English language skills

if you don't feel confident with your English.

- Talk to English Connect: https://www.monash.edu/english-connect

8.2. Study skills

If you feel like you just don't have enough time to do everything you need to, maybe you just need a new approach.

- Talk to a learning skills advisor: https://www.monash.edu/library/skills/contacts

8.3. Things are tough right now

Everyone needs to talk to someone at some point in their life, no judgement here.

- Talk to a counsellor: https://www.monash.edu/health/counselling/appointments (friendly, approachable, confidential, free)

8.4. Things in the unit don't make sense

Even if you're not quite sure what to ask about, if you're not sure you won't be alone, it's always better to ask.

- Ask in Ed
- Attend a consultation

8.5. I don't know what I need

Everyone at Monash University is here to help you. If things are tough now they won't magically get better by themselves. Even if you don't exactly know, come and talk with us and we'll figure it out. We can either help you ourselves or at least point you in the right direction.