Assignment 1

Key Information

Overview

Objective

- Work collaboratively with your team
- Develop a game with a text-based interface (see tasks in next pages)
- Practice the basic programming concepts from weeks 1-5
- Download the following scaffold to populate with your solutions:

A1_Scaffold.zip

The skeleton code provides three simple functions to get you started. Please read the individual docstrings for further details.

Submission Requirements

- You will work on each task as different functions in your Ed workspace
- Your final submission must be one zip file containing your `connect4.py`, and `connectk.py` if attempted
- **One group member must make the final submission on Moodle for the whole group**
- The zip file must follow the structure below. Please ensure there are no extra files or folders

```
A1_<group_name>.zip
└─> connect4.py
└─> connectk.py (if attempted)
```

Make sure that all functions are in the file they need to be in, with appropriate function names and arguments.

It is possible to complete the assignment without importing more modules than the ones which has been imported for you (`random` and `os`). However, if you are comfortable with modules, you may import them. Make sure they are an improvement to the program, though! Furthermore, please note that ed has a restricted set of modules, which you cannot add to. Finally, be aware that your TAs may ask you about what these modules do and how you use them during the interview.

Documentation

In this assignment, all documentation is provided as part of the skeleton code. However, for future assignments you will be required to produce documentation including docstrings for your functions (and classes) as well as inline comments.

In-line comments

Unlike the documentation, you are expected to use in-line comments throughout your program to make sure it is clear to the reader. The amount of comments necessary depends on how clear the code is by itself. Therefore, a good habit is to write clear code, so that the amount of in-line comments required is reduced. This will also reduce the number of bugs created.

Comment your code as you write it rather than after reaching a final solution, as it's a lot easier to remember why certain decisions were made and the thought process behind them when the code is fresh in your mind.

Ensure your variable names are meaningful and concise so that your code is understandable at first read. When a reader is going through your program, it should read like a story that is self-explanatory, using in-line comments to explain the logic behind blocks of code and any additional information that is relevant to the functionality of the code.

Avoid repeating yourself by commenting on lines of code that are fairly self-explanatory, and instead reserve your comments to explain higher level logic, such as explaining the overall purpose behind a small block of code.

Recommended Approach

Working as a team:

- You are expected to work with all members of your team (typically of size 3 or 4).
- You are expected to attend your applied classes to meet with your team members and the TA who supervises your team.
- During the applied class, your TA will check on your progress and on how well the team is working together. This is the opportunity to raise issues. But don't throw your teammates under the bus. There is always friction in a team. Work together to smooth it out.
- You will need to meet regularly with your team and collaborate on the assignment.

Setting up your workspace:

- At the beginning of the assignment, one member needs to:
  - Create an Ed workspace for your team and share it with the other members
  - Upload the code scaffold into workspace and share workspace with the team.
- You are expected to work on the program together using the shared workspace in Ed.

Note that if all of you are already familiar with tools like git, you may use them instead. But we won't provide technical support.

Testing your code:

- The assignment is broken up into individual coding tasks.
- You can code and test using the Ed lessons.
- These lessons are **not** to submit your work, only to test your code against some basic automated checks.

Skeleton/Starter Code:

- Tasks will start with some skeleton/starter code for each method in a class.
- Make sure function and variable names match what is listed in the task; these will be needed to pass automated tests.

Academic Integrity

- Your code will be checked against all other groups with a similarity checker.
- Anything you are able to google can be easily found by the teaching team and compared against your work.
- In this assessment, you must not use generative artificial intelligence (AI) to generate any materials or content in relation to the assessment task. This includes ChatGPT.

How can I avoid academic integrity issues?

- Never copy code from anywhere. If you learn something useful online, rewrite it from scratch. It's also the best way to make sure you have understood it.
- If a fellow student asks you for a solution to a question, try to help them build their solution. Do **not** give them yours.
- Giving your solution is just as much of an Academic Integrity breach as receiving it.
- If you feel like you physically cannot submit the assignment on time, please submit an extension request, and seek help.
- For extensions, see Moodle AU or Moodle MA.
- For help, see Moodle AU or Moodle MA.

How will I be assessed?

Your assignment mark will come from:

1. your *team submission,*
2. your *contribution factor,*
3. your *team review*.

There will be an interview scheduled with a TA (Teaching Assistant) after the submission deadline, which will help determine criteria 1 and 2. Team review is due on the day of the submission.

1. Team submission (common)

This mark is the same for all team members and comes from a review of your submission by a TA, following a rubric.

Note that automated tests do not contribute to the mark. They are here to help you check the correctness of your programs. But a program that passes all tests can still be incorrect. Furthermore, test cases do not evaluate the readability of your code, which our TAs will assess.

2. Contribution factor (individual)

The contribution factor is individual, and based on your interview only.

After your individual interview with your tutor, you will be assigned one of five colours:

- **Green (1)**
  - Able to modify the code corresponding to a task to meet different requirements, **for all tasks**.
- **Blue (0.95)**
  - Able to **modify the code** to meet different requirements, for one task. The task is **chosen by the student** among the tasks attempted by the group that are **worth the most number of points** (i.e. Tasks 6, 10, 11, 13 if they are completed).
  - Able to describe how given changes to the code affect the output, **for all tasks**.
- **Orange (0.9)**
  - Able to describe how given changes to the code affect the output, for one task (**chosen by the student**).
- **Red (0.15)**
  - Insufficient evidence
- **Black (0.1)**
  - If you do not attend the interview. Note that you can apply for special consideration (see

how to apply for a short extension on Moodle).

3. Team review

A Moodle activity called FeedbackFruits will allow you to reflect on the assignment and give anonymous feedback to your teammates. Doing the team review will give a 2% bonus to your assignment (capped at 100%).

What happens during the individual part of the interview?

This is the procedure that your TA will follow during the individual part of the interview:

1. Your TA will make sure that the interview is being recorded.
2. Both TA and student turn on their camera.
3. Your TA will check your student ID via camera.
4. Share the full screen (not the application) via zoom.
5. There will be 1 or 2 questions, depending on how fast you can solve the first one. You will not be able to refer to the programs you have submitted. Instead, we will ask you to create similar programs from scratch or from a scaffold.
6. Pick a Task to get quizzed on from among Tasks 6, 10, 11, or 13. If you have not attempted any of these questions, then and only then you can pick a different Task.
7. Your TA will give you a first programming question to answer in an ed workspace (still sharing your full screen).
8. If you do not answer the question within the interview time, then the interview is over. You will not be asked a second question.
9. If there is time left, your TA will give you a second programming question, again to answer in an ed workspace. The second question will be on a different task chosen by your TA.
10. The interview ends when you run out of time, or if your TA tells you that you have answered the second question satisfactorily.

You are not allowed to communicate to other students the questions that were asked to you. This would be a breach of academic integrity.

Effective team size and team size factor

Some teams may have fewer than 4 members. Furthermore, some teammates might be inactive. So the *effective size* of your team might not be 4. Your TA will determine this with evidence such as the team chat. If the effective team size is:

- 4, then the team size factor is 100%.
- 3, then the team size factor is 110%.

- 2, then the team size factor is 120%.
- 1, then the team size factor is 130%.

Your mark will be multiplied by this factor. Inactive students won't receive this factor.

Team of 1 or 2

Your TA will try to merge teams in order to make teams of size 3 or 4.

Teams of 1 or 2 will only occur if some students in a team drop out (officially or not).

Late penalties

Up to 7 days

- 10% lost per calendar day (or part thereof).

More than 7 days

- Zero (0) marks with no feedback given.

See on Moodle AU or Moodle MA for how to apply for special consideration.

Overall mark (individual)

Your overall mark for the assignment will be computed according to the program below.

```
max_mark = 100

team_submission_mark = float(input("Team submission mark [0, {}] = ".format(max_mark)))
contribution_factor = float(input("Contribution factor {0.1, 0.15, 0.9, 0.95, 1}= "))
effective_team_size = int(input("Effective team size {0, 1, 2, 3, 4} = "))
team_review_done = bool(int(input("Has done team review {0, 1} = ")))
late_days = int(input("Days late {0, 1, ...} = "))

#note that we are doing no data validation

possible_team_size_factors = [1.00, 1.30, 1.20, 1.10, 1.00]
if contribution_factor >= 0.9:
    team_size_factor = possible_team_size_factors[effective_team_size]
else:
    #Inactive students do not get
    #the effective team size bonus
    team_size_factor = 1

team_review_mark = 2*int(team_review_done)

mark = min(max_mark,\
team_submission_mark * contribution_factor * team_size_factor + team_review_mark)
```

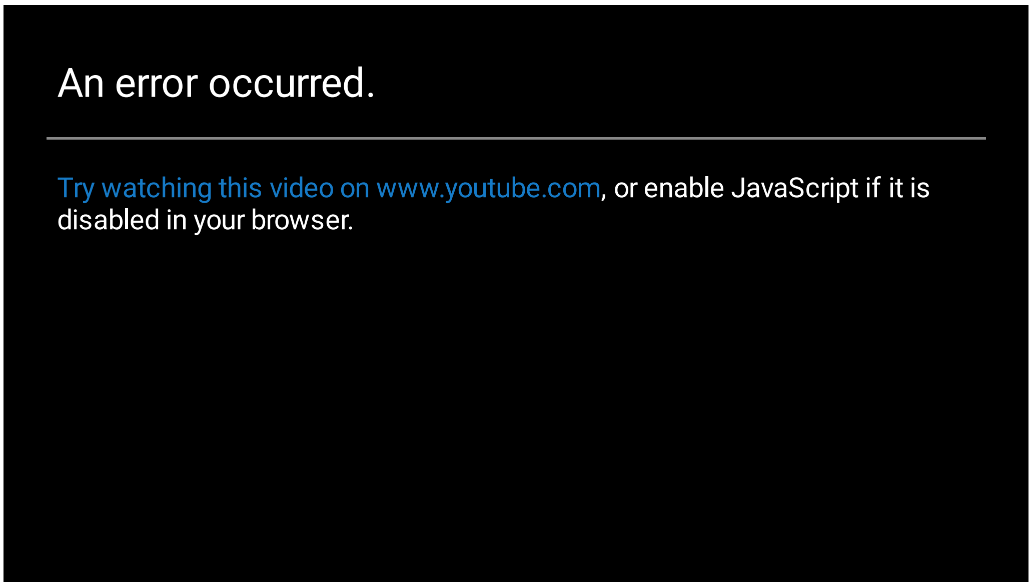
```
if late_days <= 7:
    late_penalty = 10 * late_days
else:
    late_penalty = max_mark

final_mark = max(0, mark - late_penalty)

print("The final mark is", final_mark)
```

Connect4

In this assignment, you'll be implementing a command-line version of the popular board game Connect4.

Connect4 is a two player game where players take turns dropping their counter into one of 7 columns. The game board consists of 6 rows and 7 columns, and the aim of the game is to line 4 of your pieces up either vertically, horizontally or diagonally before your opponent.

You will code the full set of functionality throughout the tasks, one part at a time.


An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

Working with functions

Each task in this assignment consists of writing your code within a predefined function scaffold.

While you will learn functions in depth during week 4, basic knowledge of how to write your code within functions will be required in order to complete this assignment.

When working in the function scaffold, you write your code as if there were no function involved, with the exception that you need to ensure the code is indented so that it is contained within the function (as you need to do in an `if` or `while`).

The basic structure of the function scaffolds provided is as follows:

```
def function():
    print("Hello, function is being executed.")
    # other code here, indented by one tab so that
    # the interpreter knows it is part of this function

if __name__ == "__main__":
    function()
```

The `if __name__ == "__main__":` part on line 7 tells the Python interpreter to execute whatever code is indented below when the file is executed. In this case we simply call the function defined above so that it's executed when the file is run.

This is identical to below:

```
 print("Hello, function is being executed.")
```

Arguments

Some of the functions provided as part of the assignment scaffold include arguments. Arguments are basically variables that the function expects when it is called.

If we have a solution to calculate the area of a circle based on its radius:

```
radius = 4
area = 3.14 * (radius ** 2)
print(area)
```

We can define it within a function that expects the radius to be passed as an argument as below:

```
def area_of_circle(radius):
    area = 3.14 * (radius ** 2)
    print(area)
```

```
if __name__ == "__main__":
    area_of_circle(4) # We define the value of the argument when calling the function
```

The tasks for this assignment will expect you to create programs that use the arguments defined to produce a result.

Returns

Many of the tasks in this assignment will expect you to return a certain value at the end of your solution.

For the example above, if we wanted the function to return the calculated area rather than print it to console, we would do:

```
def area_of_circle(radius):
    area = 3.14 * (radius ** 2)
    return area

if __name__ == "__main__":
    print(area_of_circle(4))
```

This way, the function itself simply computes a value and returns the result, while the line that calls the function can then decide what to do with the returned value (in this case also simply printing it).

Task 1: validate_input

Task Description

Create a function `validate_input` that repeatedly prompts the user for input until they enter an input within a set of valid options.

Arguments

The function takes two arguments:

- `prompt`: a string to print when asking user for input
- `valid_inputs`: a list of strings representing all options the user is restricted to

Outputs/Returns

The function should print `prompt` when asking the user for input. Upon an invalid input, please display the following message before prompting the user again: `"Invalid input, please try again."`

Once a valid input is provided, the function should return it as a string.

Example

```
>>> validate_input("Please select an option (a, b, c): ", ["a", "b", "c"])
Please select an option (a, b, c): 1
Invalid input, please try again.
Please select an option (a, b, c): 3
Invalid input, please try again.
Please select an option (a, b, c): d
Invalid input, please try again.
Please select an option (a, b, c): a
```

Task 2: create_board

Task Description

Create a function `create_board` that generates a list of lists to represent the game board for Connect4.

The board should have 6 rows and 7 columns. Each cell in the board should be initialized with a value of 0.

The length of the outer list should be 6 while the length of inner lists should be 7.

Arguments

This function takes no arguments.

Outputs/Returns

This function should return a list of lists of 6x7 dimensions.

Example

```
>>> print(create_board())
[[0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0], [0, 0,
```

Task 3: print_board

Task Description

Create a function `print_board` that prints a visual representation of the game board to console.

Arguments

This function takes a single argument:

- `board`: a 2D array

The array `board` has one of three possible integers in each cell:

- `0` for an empty cell
- `1` representing player 1's token
- `2` representing player 2's token

The top row has index 0 (and so the bottom one has index 5).

Outputs/Returns

This function should not return anything.

It should output to the console by printing multiple lines of text to create a visual representation as shown below.

```
========== Connect4 =========
Player 1: X       Player 2: O

  1   2   3   4   5   6   7
 --- --- --- --- --- --- ---
|   |   |   |   |   |   |   |
 --- --- --- --- --- --- ---
|   |   |   |   |   |   |   |
 --- --- --- --- --- --- ---
|   |   |   |   |   |   |   |
 --- --- --- --- --- --- ---
|   |   |   |   |   |   |   |
 --- --- --- --- --- --- ---
| O | X |   |   |   |   |   |
 --- --- --- --- --- --- ---
| X | X | O |   |   | O |   |
 --- --- --- --- --- --- ---
=============================
```

Each empty cell in the board should be represented by a character surrounded by a space on each side. For an empty cell, the middle character should also be a space. For a `1` or `2` the middle character should be an `X` or `O` respectively (capital 'o').

> Please be careful to not put extra spaces at the end of a line as the test cases will expect your strings to be *exact* matches! You can also use tools such as this to compare your string outputs.

Dependencies

This function depends on the completion `create_board` from task 2.

Example

```
>>> board = create_board()
>>> board[5][0], board[5][1] = 1, 2 # place a 1 in bottom left cell and a 2 in the cell directly to
>>> print_board(board)
========== Connect4 =========
Player 1: X       Player 2: O

  1   2   3   4   5   6   7
 --- --- --- --- --- --- ---
|   |   |   |   |   |   |   |
 --- --- --- --- --- --- ---
|   |   |   |   |   |   |   |
 --- --- --- --- --- --- ---
|   |   |   |   |   |   |   |
 --- --- --- --- --- --- ---
|   |   |   |   |   |   |   |
 --- --- --- --- --- --- ---
|   |   |   |   |   |   |   |
 --- --- --- --- --- --- ---
| X | O |   |   |   |   |   |
 --- --- --- --- --- --- ---
=============================
```

Task 4: drop_piece

Task Description

Create a function `drop_piece` that simulates a player dropping their piece into a column of the game board.

When dropping a piece into the selected column, the player token needs to be placed into the bottom-most empty cell of that column. Like with the real board, a player's token falls down the column and sits on top of any tokens already in that column. This means that the board should be updated if the drop is possible.

Arguments

This function takes two arguments:

- `board`: a 2D array
- `player`: an integer representing the player's token, 1 or 2
- `column`: an integer representing the column chosen by the player, 1- 7

You may assume that the column index passed will be 1-7 incl. as a player will be forced to choose one of the column labels.

Outputs/Returns

This function should return a boolean value, `True` if the chosen play was successful, or `False` if the piece cannot be dropped into the given column because it is already full.

Dependencies

This function depends on the completion `create_board` from task 2.

Example

```
>>> board = [
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 2, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 1, 2, 0, 0],
    [0, 0, 0, 2, 1, 0, 0],
    [0, 0, 0, 1, 2, 0, 0]
]
>>> print(drop_piece(board, 1, 4)
True
```

```
>>> board = [
    [0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 2, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 1, 2, 0, 0],
    [0, 0, 0, 2, 1, 0, 0],
    [0, 0, 0, 1, 2, 0, 0]
]
>>> print(drop_piece(board, 2, 4)
False
```

Task 5: execute_player_turn

Task Description

Create a function `execute_player_turn` that prompts the user for a legal move given the game board.

Use your solution from task 1 to pass a prompt (see example below) for the user to select a column, and the list of acceptable inputs: `["1", "2", "3", "4", "5", "6", "7"]`

This function should then try to drop a piece into the column selected. If the drop is unsuccessful, it should print a message such as `"That column is full, please try again."` and prompt the user for input again.

Arguments

This function takes a two arguments:

- `player`: an integer representing the player's piece, 1 or 2
- `board`: a 2D array of 6x7 dimensions

Outputs/Returns

This function should prompt the user by printing a message to select their desired column.

If the user's selected drop is not successful, the function should print a message (see example below) saying so before prompting the user again.

This function returns the player's chosen column as an integer once a successful drop is confirmed.

Dependencies

This function depends on the completion of `validate_input` from task 1, `create_board` from task 2 and `drop_piece` from task 4.

Example

```
>>> board = [
    [0, 0, 0, 0, 2, 0, 0],
    [0, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 1, 2, 0, 0],
    [0, 0, 0, 1, 2, 0, 0],
    [0, 0, 0, 2, 1, 0, 0],
    [0, 2, 1, 1, 2, 0, 0]
]
>>> print(execute_player_turn(1, board))
Player 1, please enter the column you would like to drop your piece into: 5
That column is full, please try again.
Player 1, please enter the column you would like to drop your piece into: 4
4
```

End of Week 3 Checkpoint

We recommend you complete at least Tasks 1-5 during week 3.

Your TA will check the progress of your team during your applied class.

Task 6: end_of_game

Task Description

Create a function `end_of_game` that checks the current state of the game board.

It should check whether a player has won, whether the board is full, or whether the game should continue.

A win means 4 connected tokens either vertically, horizontally or diagonally (in either direction).

Please note you are welcome to create your own helper functions in order to implement this function.

Arguments

This function takes a single argument:

- `board`: a 2D array of 6x7 dimensions

Outputs/Returns

The function should return an integer value that represents the current state of the game:

  - `0` if the game is not over
  - `1` if player 1 wins
  - `2` if player 2 wins
  - `3` if the game is over and it's a draw

You may assume that the board passed will always be a valid game board (i.e. no floating pieces).

For the game to be considered a draw, there must be no available space in the game board and neither player has managed to line up 4 pieces.

Example

```
>>> board = [
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0],
    [1, 2, 2, 2, 0, 0, 0]
]
>>> print(end_of_game(board))
```

```
0
```

```
>>> board = [
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0],
    [1, 2, 2, 2, 0, 0, 0]
]
>>> print(end_of_game(board))
1
```

```
>>> board = [
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0],
    [1, 2, 2, 2, 2, 0, 1]
]
>>> print(end_of_game(board))
2
```

```
>>> board = [
    [1, 2, 1, 1, 2, 2, 1],
    [2, 1, 2, 2, 1, 2, 2],
    [2, 1, 2, 2, 1, 2, 1],
    [1, 1, 1, 2, 1, 1, 1],
    [1, 2, 2, 1, 2, 1, 2],
    [2, 2, 2, 1, 2, 1, 2]
]
>>> print(end_of_game(board))
3
```

Task 7: local_2_player_game

Task Description

Create a function `local_2_player_game` that runs a regular game of Connect4.

This function should initialise a new game board and execute a loop where each iteration represents one turn of the game.

Game loop logic:

The first turn should be set to player 1.

During each iteration:

- Clear the console by calling `clear_screen`.
- Print the game board.
- Display the previous player and their chosen move if there is one.
- Execute the current player's turn.
- Check if the game has ended.

The game loop should terminate once the game has ended, after which the final state of the board should be printed followed by the result (there are no prescribed strings for printing the game result).

Please note that there are no test cases for this task (hence no 'Submit' button). In order to test your solution for this task you will need to run the code.

Arguments

This function takes no arguments.

Outputs/Returns

This function has no returns.

Dependencies

This function requires the completion of all prior tasks.

Example

```
========== Connect4 =========
Player 1: X       Player 2: O
```

```
  1   2   3   4   5   6   7
 --- --- --- --- --- --- ---
|   |   |   |   |   |   |   |
 --- --- --- --- --- --- ---
|   |   |   |   |   |   |   |
 --- --- --- --- --- --- ---
|   |   |   |   |   |   |   |
 --- --- --- --- --- --- ---
|   |   |   |   |   |   |   |
 --- --- --- --- --- --- ---
|   |   |   |   |   |   |   |
 --- --- --- --- --- --- ---
| O | X | X |   |   |   |   |
 --- --- --- --- --- --- ---
=============================
Player 1 dropped a piece into column 2
Player 2, please enter the column you would like to drop your piece into:
```

Task 8: main

Task Description

Create a function `main` that executes the main menu loop for our application.

The loop should execute until a user choses to exit the program.

Call `clear_screen` to clear the console before displaying a menu screen.

Prompt the user to select one of four options:

- View rules
- Play a local 2 player game
- Play a game against the computer
- Exit

In order to display rules, clear the screen again and call `print_rules`.

With the option to play a game against the computer, you can call `game_against_cpu` however it will raise a `NotImplementedError` until you implement it.

Please note that there are no test cases for this task (hence no 'Submit' button). In order to test your solution for this task you will need to run the code.

Arguments

This function takes no arguments.

Outputs/Returns

This function has no returns.

The function should print a menu screen for the user, for example:

```
=============== Main Menu ===============
Welcome to Connect 4!
1. View Rules
2. Play a local 2 player game
3. Play a game against the computer
4. Exit
=========================================
```

Dependencies

This function requires the completion of all prior tasks.

Task 9: cpu_player_easy

Task Description

Create a function `cpu_player_easy` that represents the computer player using a simple strategy to play its turn during a game.

It should repeatedly try to drop a piece into the board in a randomly selected column until a successful drop is recorded.

The random module has been imported for you in the scaffold for this function. To generate a random integer between `x` and `y` inclusive, call `random.randint(x, y)`.

Arguments

This function takes a two arguments:

- `board`: a 2D array of 6x7 dimensions
- `player`: an integer representing the player's piece, 1 or 2

Outputs/Returns

This function returns the column chosen upon a successful drop as an integer. Please note that this column index must be 1 based.

Dependencies

This function depends on the completion of `drop_piece` from task 4.

Example

```
>>> board = [
[0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[1, 1, 0, 0, 0, 0, 0],
[1, 2, 2, 2, 0, 0, 1]
]
>>> print(cpu_player_easy(board, 2))
1
>>> for row in board: print(row)
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
```

```
[0, 0, 0, 0, 0, 0, 0]
[2, 0, 0, 0, 0, 0, 0]
[1, 1, 0, 0, 0, 0, 0]
[1, 2, 2, 2, 0, 0, 1]
```

End of Week 4 Checkpoint

We recommend you complete at least Tasks 6-9 during week 4.

Your TA will check the progress of your team during your applied class.

Task 10: cpu_player_medium

Task Description

Create a function `cpu_player_medium` that represents the computer player using a slightly more advanced strategy to play its turn during a game.

It should analyse the game board to see if it can play a move that results in an immediate win. If there is such a move, it should play it. If not, it should check if the opponent can score an immediate win, and block it.

⚠ Taking an immediate win should be prioritised over blocking an immediate win.

If neither of those options are available, it should try a randomly selected column until a successful drop is recorded.

ℹ Please note you are welcome to create your own helper functions in order to implement this function.

Arguments

This function takes a two arguments:

- `board`: a 2D array of 6x7 dimensions
- `player`: an integer representing the player's piece, 1 or 2

Outputs/Returns

This function returns the column chosen upon a successful drop as an integer. Note that this column index must be 1 based.

Dependencies

You may use functions that you have written in previous tasks.

Example

```
>>> board = [
[0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[1, 1, 0, 0, 0, 0, 0],
[1, 2, 2, 2, 0, 0, 1]
]
>>> print(cpu_player_medium(board, 2))
5
>>> for row in board: print(row)
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[1, 1, 0, 0, 0, 0, 0]
[1, 2, 2, 2, 2, 0, 1]
```

Task 11: cpu_player_hard

Task Description

Create a function `cpu_player_hard` that represents the computer player using an even more advanced strategy.

For this task it is up to you to come up with a strategy as long as it represents a stronger player than `cpu_player_medium`. The automated tests will put your implementation of `cpu_player_hard` against the solution for `cpu_player_medium` in 100 simulated games of Connect4. The performance of your player determines the marks earned for this task (please refer to marking rubric for more details).

In the docstring for this function you must also include documentation explaining your strategy.

Arguments

This function takes a two arguments:

- `board`: a 2D array of 6x7 dimensions
- `player`: an integer representing the player's piece, 1 or 2

Outputs/Returns

This function returns the column chosen upon a successful drop as an integer. Note that this column index must be 1 based.

Task 12: game_against_cpu

Task Description

Create a function `game_against_cpu` that runs a regular game of Connect4.

This function should first prompt the user to select a difficulty level between the three options of easy, medium and hard CPU players. The selected difficulty should influence which of the three CPU player functions is called during the game.

The function should then initialise a new game board and execute a loop where each iteration represents one turn of the game.

Please note that there are no test cases for this task (hence no 'Submit' button). In order to test your solution for this task you will need to run the code.

Game loop logic:

The first turn should be set to player 1.

During each iteration:

- Clear the console by calling `clear_screen`.
- Print the game board.
- Display the previous move for *both* the CPU player and the user (except for the first round).
- Execute the current player's turn.
- Check if the game has ended.

The game loop should terminate once the game has ended, after which the final state of the board should be printed followed by the result.

Arguments

This function takes no arguments.

Outputs/Returns

This function has no returns.

Dependencies

This function requires the completion of all prior tasks.

Task 13: Connectk

Task Description

Create a separate file `connectk.py` and program a more general version of connect4. At the start of a game of Connectk, the user inputs:

- The number of rows of the game board,
- The number of columns of the game board,
- The number k of tokens that need to be connected in order to win,
- The number of human players,
- The number and level of CPU players.

In this version, the total number of players is not limited to 2.

In order to complete this task, we ask you to rewrite all functions from Tasks 1 to 12 for Connectk. This will involve modifying not just the functions, but their arguments and return values.

✕ This task is much harder than the previous ones. It is designed to provide an extra challenge to "HD students". If you have completed everything up until this point but have other assignments to work on, consider not attempting this task.

End of Week 5 Checkpoint

Ideally, you will have completed Tasks 1-12 (and even perhaps Task 13) by the end of Week 5.

Your TA will check the progress of your team during your applied class.

Checklist for submission day

Please check that you have done all the following by submission day:

1. The work has been submitted according to the instructions given in the overview slide.
2. You and your teammates have picked an interview slot (TBA). Also make sure that:
    1. The interview is with your team tutor.
    2. Only one team member has booked the interview.
3. You have submitted team feedback and self-reflection via the activity on the assessment page (Moodle AU or Moodle MA).

Any deviation from the rule 1 and 2 will incur a 10% penalty (same a late penalty).

Marking Rubric

Changelog - 12 changes

13/03/2023 - Change 1

- Specified which tasks students need to choose from for the colour "Blue" of the interview.
- In Task 2, changed references of "2D array" to "list of lists" for clarity.

15/03/2023 - Change 2

Updated docstring for cpu_player_hard to include the correct return value as per task description (previous None).

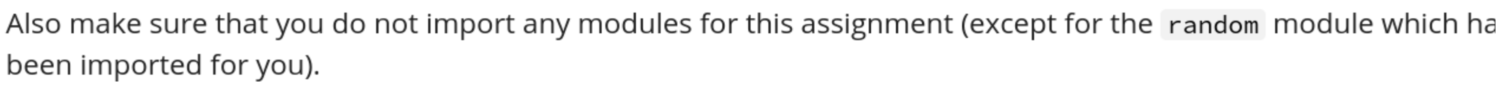
16/03/2023 - Change 3

Added this description to Task 6. It does not change how the task is meant to be completed, but it provides some details that may not have been immediately clear.

> It should check whether a player has won, whether the board is full, or whether the game should continue.
>
> A win means 4 connected tokens either vertically, horizontally or diagonally (in either direction).

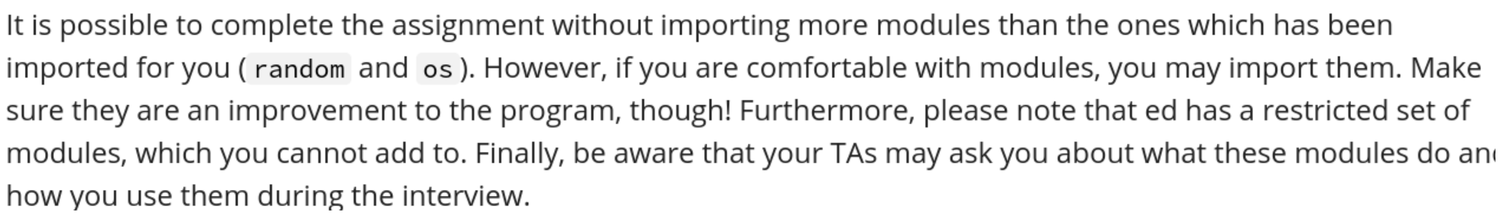
17/03/2023 - Change 4

We now authorise you to use modules. In the slide Overview, we changed this text:



Also make sure that you do not import any modules for this assignment (except for the `random` module which ha been imported for you).

to this text



It is possible to complete the assignment without importing more modules than the ones which has been imported for you (`random` and `os`). However, if you are comfortable with modules, you may import them. Make sure they are an improvement to the program, though! Furthermore, please note that ed has a restricted set of modules, which you cannot add to. Finally, be aware that your TAs may ask you about what these modules do an how you use them during the interview.

This is because some students made a good case why they could be useful in this assignment.

17/03/2023 - Change 5

Task 10 had an example with an error. This was the erroneous example:

```
>>> board = [
[0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[1, 1, 0, 0, 0, 0, 0],
[1, 2, 2, 2, 0, 0, 1]
]
>>> print(cpu_player_medium(board, 2))
1
>>> for row in board: print(row)
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[1, 1, 0, 0, 0, 0, 0]
[1, 2, 2, 2, 2, 0, 1]
```

However the cpu should play in column 5. So we have fixed this to be:

```
>>> board = [
[0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[1, 1, 0, 0, 0, 0, 0],
[1, 2, 2, 2, 0, 0, 1]
]
>>> print(cpu_player_medium(board, 2))
5
>>> for row in board: print(row)
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[1, 1, 0, 0, 0, 0, 0]
[1, 2, 2, 2, 2, 0, 1]
```

Thanks Gia for reporting this!

18/03/2023 - Change 6

- Added test cases for task 10 and 11 to ensure only one token is dropped.

23/03/2023 - Change 7

Changed the rubric for task clarify the return value for task 1. Thank you Harrison for pointing this out.

30/03/2023 - Change 8

Changes to task 4 test cases

31/03/2023 - Change 9

Changes to task 10 test cases.

01/04/2023 - Change 10

Updated the section "What happens during the individual part of the interview?" in the slide How will

I be assessed?.

02/04/2023 - Change 11

Updated the section "What happens during the individual part of the interview?" in the slide How will I be assessed? by adding:

> You will not be able to refer to the programs you have submitted. Instead, we will ask you to create similar programs from scratch or from a scaffold.

04/04/2023 - Change 12

Updated the section "What happens during the individual part of the interview?" in the slide How will I be assessed? by adding:

> The second question will be on a different task chosen by your TA.