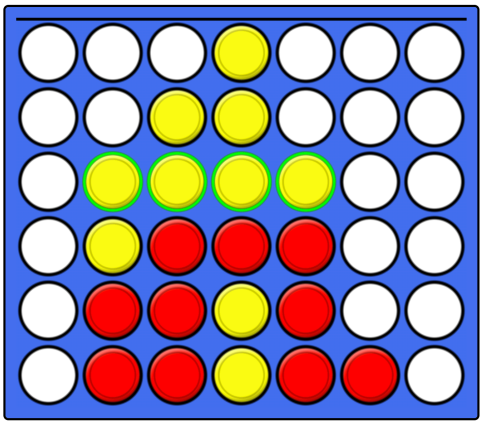
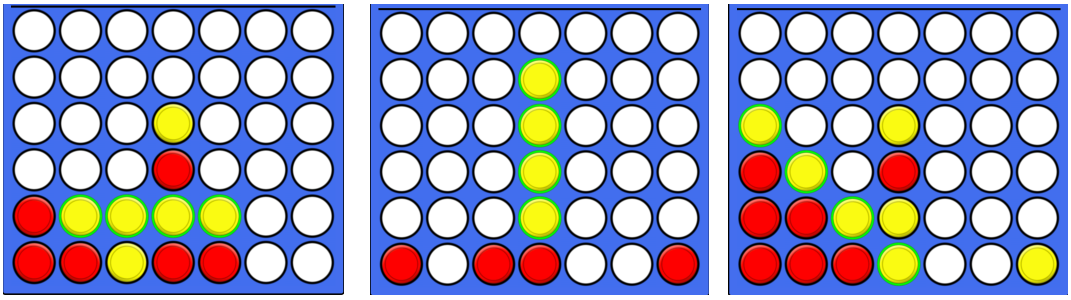
# Connect 4 - 2023



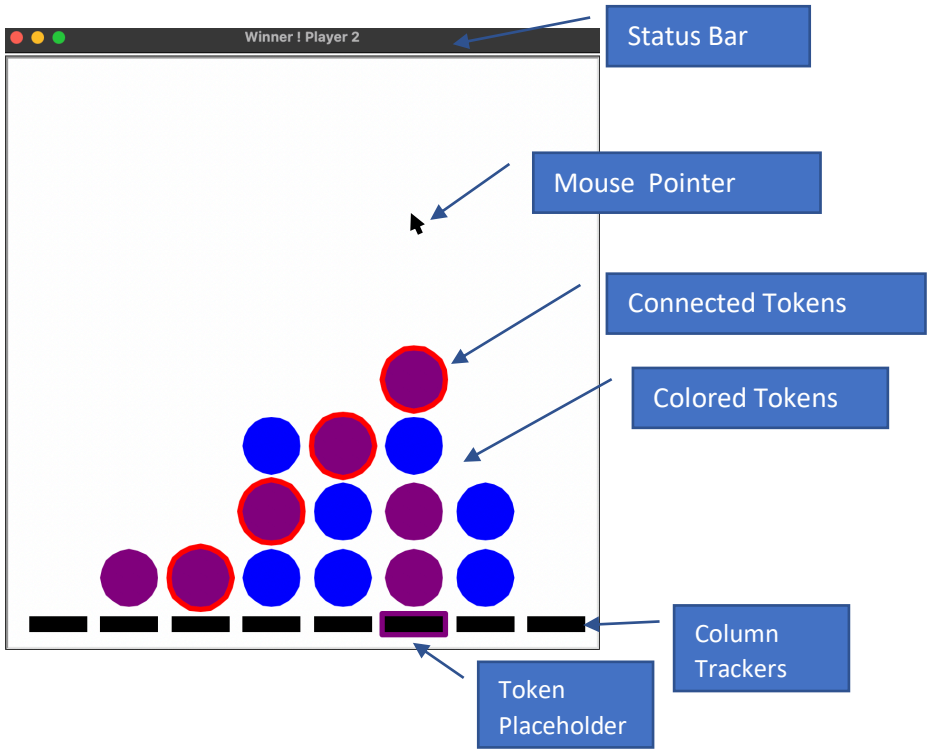
## OVERVIEW

In this assignment, you are going to design and to develop a 2-player connection board game, in which the players first choose a color called token and then take turns dropping the colored token into columns, and the first player who connects four tokens of same color before the other opponent does wins the game. The four connected tokens must be placed in four consecutive positions in a row, be it horizontally, vertically or diagonally (left and right). The following 3 diagrams show 3 different winning connections (yellow tokens) in horizontal, vertical and diagonal positions respectively.



# SCOPE

1. Implement the board game using the standard Turtle Graphics module. The game board is a fixed 8-column by 8-row grid. The following diagram shows the overall layout of the game:

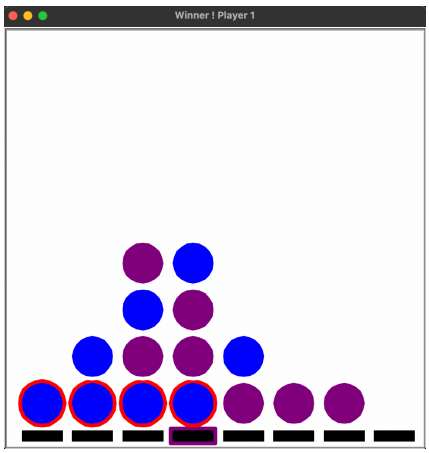



2. Status Bar
    a. It's the standard title bar of the Turtle Graphics' Windows. Your program will use it to display a simple string to show the FINAL status of the game:
        i. Winner ! Player 1, or
        ii. Winner ! Player 2, or
        iii. Game Tied !

        Note: during the game you can simply display a default, static title such as "Connect 4", optionally including the player identifier, such as Connect 4 – Player 1 Turn, or Connect 4 – Player 2 Turn.

        Hint: use title() from the Screen object to set the text string in the title bar.

3. Connected Tokens
    a. At end of the game, if there is a winner, outline the 4 connected tokens with another different color (red as shown in the diagram above). Note that it's possible there are multiple winning rows such as the horizontal and diagonal rows in blue as shown in the following diagram. In this case, your program simply shows any one of the winning rows.




4. Colored Tokens

   a. Use standard shape such as circle or square to implement colored tokens and pick a different color for each player (ex: blue and purple as shown). Set both width and height of the token shape to 60 pixels. The default size in pixels for standard circle and square shapes is 20. Hint: for turtle object use function shapesize() to stretch the size accordingly.

5. Column Trackers

   a. Show the column positions of the game board. Separate column trackers with small margin in between, including to the left side of the first column and to the right of last column (see diagram above). Match the width of each column tracker to the colored token's shape, and set the height of each column tracker to 20 pixels (default size).

6. Token Placeholder

   a. Based on the current position of the mouse pointer, outline the corresponding column tracker with the color that matches that of the current player. Set the width of the outline to around 5 pixels. Hint: for turtle object you could use color() to set the outline color and use shapesize() to set the width of the outline. See "Mouse Tracking" for more info.

7. Size & Colors
   a. You are free to choose your own set of colors (fill and outline combined) provided that different colors are consistently used for different components such as color tokens, column tracker, token placeholder, connected tokens and so on.
   b. Stick to the measurement (pixel size) suggested for tokens and column trackers, or choose a different measurement appropriately.

8. Token Dropping

   a. Mouse click is used to initiate the action of dropping a token to the currently outlined column (token placeholder). It's not mandatory to implement the logic to show the dropping of the token from the top of the column. In other words, your program can instantly and directly display the token in the column at the correct position right after the mouse click.

9. Mouse Tracking
   a. You are only required to track the mouse pointer when it's hovered within the Turtle Graphics' window. Therefore, as the mouse pointer hovers out of the Turtle Graphics' window, leave the outlined column as it is. As soon as the mouse pointer is within the Turtle Graphics' window, your tracking logic should outline the column correspondingly.
   b. Hint: use bind() function from the canvas object to bind the mouse motion to a separate function. To get the canvas object use getcanvas() from the turtle module.
   c. Hint: Note that you need to implement timer logic with a separate function to perform the tracking logic as mentioned above. See "Timer - Screen Refresh" for more info.
10. Timer – Screen Refresh
   a. In general, if your logic relies on a timer to perform repeated computation at a regular interval ensure that the value of timer rate is set appropriately. For an example, if timer is used for "Mouse Tracking", you would set a small value for the timer rate, such as 100 msec. If it's set to a higher timer rate, say 2 seconds (2000 msec), the outlined column position would not accurately match the mouse pointer's position when it's in motion.

NOTE:

- Keep your entire source code in ONE SINGLE file.
- Use only standard python modules
- In your design stick ONLY to functions, in other words, no class objects of your own.

# STARTUP OPTIONS

Not applicable

# SKILLS

In this assignment, you will be trained on the use of the followings:

- Logic Decomposition: Functions (with parameters and return) for program structure and logic decomposition
- GUI based programming – turtle graphics
    - Event, Timer, Mouse Tracking, and so on.
- Standard objects (strings, numbers & lists)
- Variable Scope

# DELIVERABLES

1. Program source code (A2_School_StudentID_Source.py)

where School is SSE, SDS, SME, HSS, FE or LHS and StudentID is your 9-digit student ID.

Submit the python file by due date to the corresponding assignment folder under "Assignment (submission)"

For instances, a SME student with student ID "119010001" will name the program as follows:

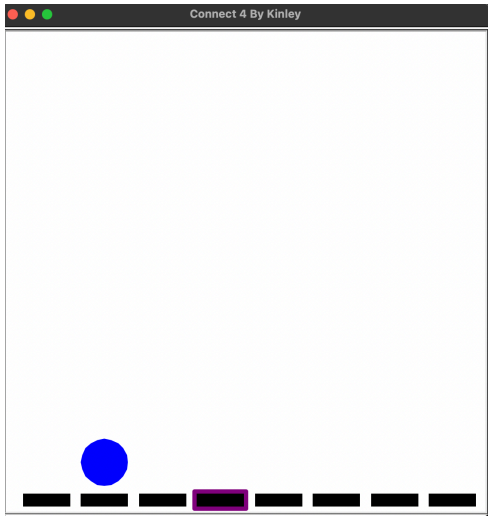
- A2_SME_119010001_Source.py:

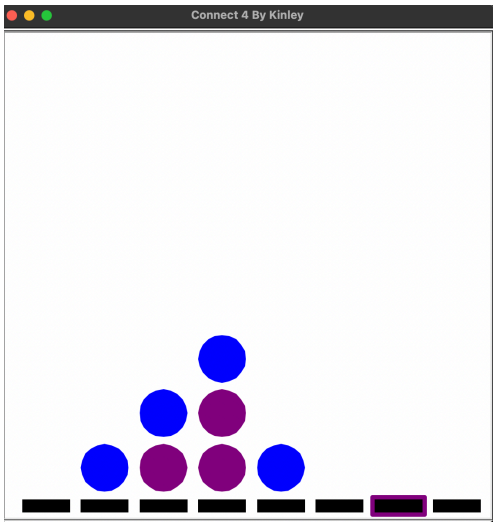
5% will be deducted if file is incorrectly named!!!

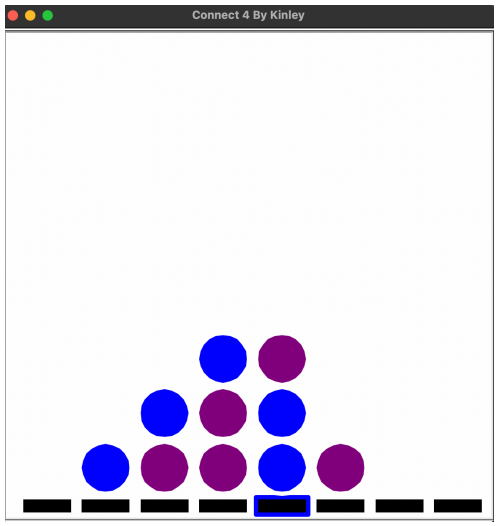
# TIPS & HINTS

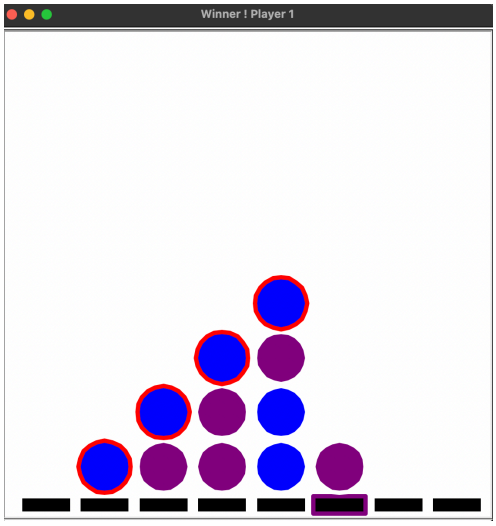
- Beware of variable scope as you might keep a few variables as global such as puzzle
- Refer to python website for program styles and naming convention (PEP 8)
- Validate all inputs (if any) - if incorrect input is detected then display a friendly response and prompt the input again.
- Use Canvas to bind the motion of the mouse pointer to your program
    - The x,y coordinate is upper-left corner based (0,0) and window coordinate.
- Use separate timer to update the column tracker
- Use regular turtle objects as tokens and column trackers.
- Use color() function to change the fill and outline colors of the turtle objects.
- Use ontimer() function to perform repeated computation at a regular time interval in msec.
- Use shapesize() function to change the size of the turtle objects.
- Use setworldcoordinates() function to change the x-y coordinates of the lower left and upper right corners. All drawing follows world coordinate setting. Hint: best to keep the same resolution as the Turtle Graphics' Window (screen size).

# SAMPLE OUTPUT

# MARKING CRITERIA

- Coding Styles – overall program structure including layout, comments, white spaces, naming convention, variables, indentation, functions with appropriate parameters and return.
- Design Documentation if required
- Program Correctness – whether or the program works 100% as per Scope.
- User Interaction – how informative and accurate information is exchanged between your program and the player.
- Readability counts – programs that are well structured and easy-to-follow using functions to breakdown complex problems into smaller cleaner generalized functions are preferred over a function embracing a complex logic with nested conditions and sub-functions! In other words, a design with clean architecture with high readability is the predilection for the course objectives over efficiency. The logic in each function should be kept simple and short, and it should be designed to perform a single task and be generalized with parameters as needed.
- KISS approach – Keep It Simple and Straightforward.
- Balance approach – you are not required to come up a very optimized solution. However, take a balance between readability and efficiency with good use of program constructs.

| ITEMS | PERCENTAGE | REMARKS |
|---|---|---|
| CODING STYLES | 15%-20% | 0% IF PROGRAM DOESN'T RUN |
| USER INTERFACE | 15%-20% | 0% IF PROGRAM DOESN'T RUN |
| FUNCTIONALITY | MAX. 70% | REFER TO SCOPE |

# DUE DATE

Apr 2nd, 2023, 11:59:59PM