Faculty Of Computers and Artificial Intelligence
Cairo University

CS213: Programming II
Year 2022-2023
First Semester

Assignment 3 – Version 2.0

Course Instructors:
Dr. Mohammad El-Ramly

Name	ID
Shahd Mohamed Ahmed Sayed	20220533
Basma Hany Abdel Moneim	20220083
Nada Ahmed Sayed Basuony	20220357

- **Divided Task_1**

Basma_20220083	P1 , P4
Nada_20220357	P2 , P5
Shahd_20220533	P3 , P6

- **Divided Task_2**

Basma_20220083	**Game_1**
Nada_20220357	**Game_2**
Shahd_20220533	**Game_3**

- **Algorithms of Task_2**

1. **Game_1:**
   pyramid_X_O_Board class:

   Initialization Algorithm (pyramid_X_O_Board constructor):
   Set n_rows to 3 (Three rows for a pyramid).
   Set n_cols to 5 (Five columns for the base of the pyramid).
   Allocate memory for a 2D array (board) of size n_rows x n_cols.
   Initialize all elements of the board to 0.
   Update Board Algorithm (update_board function):
   Check if the move is within valid boundaries (not outside the board, and not on specific prohibited positions).

Valid boundaries: 0 <= x <= 2, 0 <= y <= 4
Prohibited positions: (0,2), (1,0), (1,4)
Also, check if the chosen position is empty (board[x][y] == 0).
If the move is valid, update the board at position (x, y) with the uppercase of the provided mark.
Increment the n_moves counter.
Return true to indicate a successful update; otherwise, return false.
Display Board Algorithm (display_board function):
Print the visual representation of the board with labeled positions.
Print the first row: | (0,2) |
Print the second row: | (1,1) | (1,2) | (1,3) |
Print the third row: | (2,0) | (2,1) | (2,2) | (2,3) | (2,4) |
Winning Condition Algorithm (is_winner function):
Check for horizontal wins:
Check if board[1][1], board[1][2], and board[1][3] are equal and not 0.
Check if board[2][0], board[2][1], and board[2][2] are equal and not 0.
Check if board[2][1], board[2][2], and board[2][3] are equal and not 0.
Check if board[2][2], board[2][3], and board[2][4] are equal and not 0.
Check for vertical win:
Check if board[0][2], board[1][2], and board[2][2] are equal and not 0.
Check for diagonal wins:
Check if board[0][2], board[1][1], and board[2][0] are equal and not 0.
Check if board[0][2], board[1][3], and board[2][4] are equal and not 0.
Draw Condition Algorithm (is_draw function):
Return true if the total number of moves (n_moves) is 9 and there is no winner; otherwise, return false.
Game Over Condition Algorithm (game_is_over function):

Return true if the total number of moves (n_moves) is greater than or equal to 9; otherwise, return false.

2. **Game_2:**

1-Define the Board class with the following methods:

update_board(y, symbol): Updates the board with the given move at position (y, symbol).
is_winner(): Checks if there is a winner on the board.
is_draw(): Checks if the game is a draw.
display_board(): Displays the current state of the board.
game_is_over(): Checks if the game is over.
2-Define the Connect_four_Board class, which inherits from the Board class. Implement the methods based on the Connect Four game rules.

3-Define the Player class with the following methods:

get_move(y): Gets the move from the player.
to_string(): Converts the player object to a string representation.
get_symbol(): Returns the player's symbol.
4-Define the RandomPlayer class, which inherits from the Player class. Implement the get_move method to generate a random move.

5-Define the GameManager class with the following methods:

run(): Runs the game loop.
GameManager(Board*, Player* playerPtr[2]): Initializes the game manager with the board and players.
6-Implement the Connect_four_Board class methods:

Connect_four_Board(): Initializes the Connect Four board.

update_board(y, mark): Updates the board with the given move if it is valid.
is_winner(): Checks if there is a winner on the Connect Four board.
display_board(): Displays the current state of the Connect Four board.
is_draw(): Checks if the game is a draw.
game_is_over(): Checks if the game is over.
7-Implement the Player class methods:

Player(symbol): Initializes the player with the given symbol.
Player(order, symbol): Initializes the player with the given order and symbol.
get_move(y): Gets the move from the player.
to_string(): Converts the player object to a string representation.
get_symbol(): Returns the player's symbol.
8-Implement the RandomPlayer class methods:

RandomPlayer(symbol, dimension1): Initializes the random player with the given symbol and dimension.
get_move(y): Generates a random move for the random player.
9-Implement the GameManager class methods:

GameManager(bPtr, playerPtr[2]): Initializes the game manager with the board and players.
run(): Runs the game loop:
Displays the board.
Gets the move from each player.
Updates the board with the move.
Displays the updated board.
Checks if there is a winner or a draw.
If there is a winner, declares the winner and ends the game.
If it is a draw, declares a draw and ends the game.
10-Implement the main function:

```
Create an array of Player pointers.
Initialize the players with the desired options (human vs. human or human vs. computer).
Create a GameManager object with a Connect_four_Board and the players.
Run the game.
```

3. **<u>Game_3</u>**

Game3_X_O_Board class

```
1. Start
2. Declare class Game3_X_O_Board
3. Define the constructor Game3_X_O_Board():
  - Set n_rows and n_cols to 5
  - Create a dynamic 2D array called board with n_rows rows and n_cols columns
  - Initialize all elements of the board to 0
  - Set n_moves to 0
4. Define the function update_board3(int x, int y, char mark):
  - Check if x and y are within the valid range (0-4) and the position on the board is empty (board[x][y] == 0)
  - If the conditions are met:
    - Assign the uppercase of mark to board[x][y]
    - Increment n_moves by 1
    - Return true
  - Else:
    - Return false
5. Define the function display_board3():
  - Iterate over the rows i from 0 to 4:
    - Print a new line and a vertical separator "|"
    - Iterate over the columns j from 0 to 4:
      - Print the coordinates of the cell (i, j)
      - Print the value of board[i][j] with a width of 2 characters
      - Print a vertical separator "|"
    - Print a new line and a horizontal separator "-----------------------------"
  - Print a new line
6. Define the function is_winner3():
  - Declare an array of symbols Symbol[] = {'X', 'O'}
  - Declare an array to store the number of wins for each player Player_cnt[] = {0, 0}
  - Iterate over each symbol sy in Symbol[]:
    - Iterate over the rows i from 0 to n_rows-1:
      - Iterate over the columns j from 0 to n_cols-1:
        - Check if there is a winning combination in the columns, rows, diagonal, or reverse diagonal for the current symbol sy:
          - If a winning combination is found, increment the win count for the corresponding player in Player_cnt[]
  - Check if the total number of moves is equal to 24:
    - If true:
      - Check which player has more wins:
        - Print the winner and the number of times they won
        - Return true
      - If the number of wins is equal for both players:
        - Print "Draw game!"
        - Return true
  - Return false
7. Define the function is_draw3():
  - Check if the total number of moves is equal to 24 and the game is not won:
    - If true, return true
  - Otherwise, return false
8. Define the function game_is_over3():
  - Check if the total number of moves is greater than or equal to 24:
    - If true, return true
  - Otherwise, return false
9-End.
```

XO_GameManager3 class:

1. Create a class called XO_GameManager3.
2. Define the class variables and methods.

3. Create a constructor for the class that takes a Board3 object pointer (bPtr) and an array of Player3 object pointers (playerPtr) as parameters.

4. Inside the constructor, assign the bPtr to the boardPtr variable and assign playerPtr[0] and playerPtr[1] to players[0] and players[1] respectively.

5. Create a method called run() with no parameters and return type void.

6. Inside the run() method:

a. Declare two variables x and y to store the coordinates of the player's move.

b. Display the current state of the game board by calling the display_board3() method of the boardPtr object.

c. Start a loop that runs for 24 iterations (representing 12 moves for each player).

d. Inside the loop, iterate over the players array using a for-each loop:

i. Call the get_move() method of the current player to get their desired move coordinates and store them in x and y.

ii. Use a while loop to continuously prompt the player for a valid move until the update_board3() method of the boardPtr object returns true.

- Inside the while loop, call the get_move() method of the current player to get their desired move coordinates and store them in x and y.

iii. After a valid move is obtained, call the update_board3() method of the boardPtr object to update the game board with the current player's move.

iv. Display the updated game board by calling the display_board3() method of the boardPtr object.

v. Check if there is a winner on the board by calling the is_winner3() method of the boardPtr object.

vi. Check if the game is over by calling the game_is_over3() method of the boardPtr object.

- If the game is over, return from the run() method.

e. End the loop.

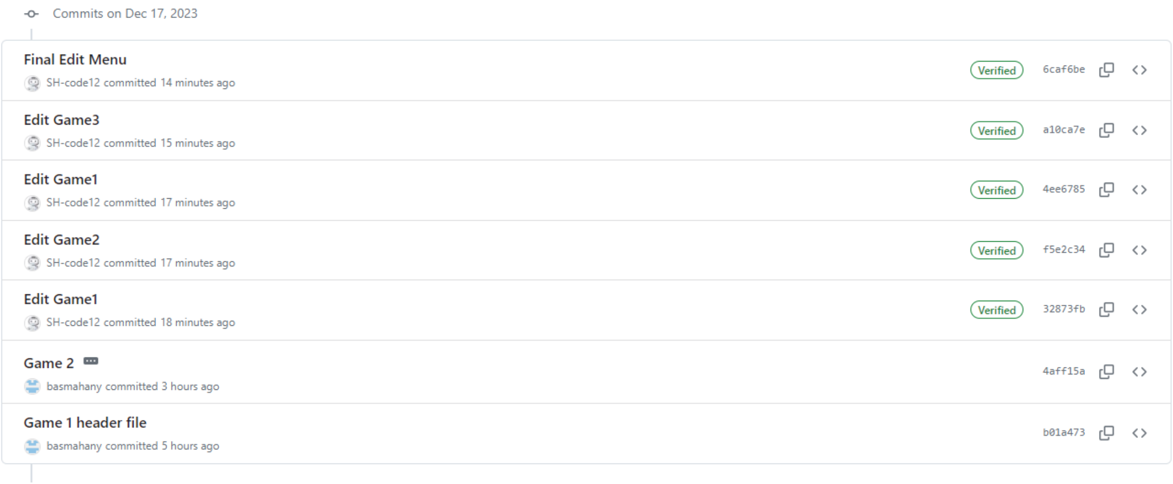
1. Import necessary libraries and header files.

```
2. Define the class Player3:
   a. Define the constructor Player3(symbol):
      i. Print "Welcome Player 1 ^_^".
      ii. Set the symbol of the player as the provided symbol.

   b. Define the method get_move(x, y):
      i. Print "Please enter your move x and y (0 to 4) separated by spaces: ".
      ii. Read the values of x and y from the user.

   c. Define the method get_symbol():
      i. Return the symbol of the player.

3. Define the main function:
   a. Create an instance of Player3 with a symbol as a parameter.

4. End of the program.
*
```

Menu:

```
START
    n = 0
    WHILE n == 0 DO
        DISPLAY "Welcome ya Ala Player ^_^"
        DISPLAY "Menu:"
        DISPLAY "(1) Traditional X_O"
        DISPLAY "(2) Pyramic Tic-Tac-Toe"
        DISPLAY "(3) Four-in-a-row"
        DISPLAY "(4) 5 x 5 Tic Tac Toe"
        DISPLAY "(5) Exit"
        READ choice

        IF choice == 1 THEN
        READ choice_player
        CREATE players[2]
        players[0] = new Player(1, 'x')

        DISPLAY "Welcome to FCAI X-O Game. :)"
        DISPLAY "Press 1 if you want to play with computer:"
        READ choice_player

        IF choice_player != 1 THEN
            players[1] = new Player('o')
        ELSE
            players[1] = new RandomPlayer('o', 5) // change dimension

        CREATE x_o_game(new X_O_Board, players)
        RUN x_o_game
        EXECUTE system("pause")

    ELSE IF choice == 2 THEN
        READ choice_player
        CREATE players[2]
        players[0] = new Player1(1, 'x')

        DISPLAY "Welcome to FCAI X-O Game. :)"
        DISPLAY "Press 1 if you want to play with computer:"
        READ choice_player

        IF choice_player != 1 THEN
            players[1] = new Player1('o')
        ELSE
            players[1] = new RandomPlayer1('o', 5) // change dimension

        CREATE x_o_game1(new pyramid_X_O_Board, players)
        RUN x_o_game1
        EXECUTE system("pause")
    ELSE IF choice == 3 THEN
        READ choice_player
        CREATE players[2]
        players[0] = new Player2(1, 'x')

        DISPLAY "Welcome to FCAI X-O Game. :)"
        DISPLAY "Press 1 if you want to play with computer:"
        READ choice_player

        IF choice_player != 1 THEN
            players[1] = new Player2('o')
        ELSE
            players[1] = new RandomPlayer2('o', 7) // change dimension

        CREATE x_o_game2(new Connect_four_Board, players)
        RUN x_o_game2
        EXECUTE system("pause")

    ELSE IF choice == 4 THEN
        READ choice_player
        CREATE players[2]
        players[0] = new Player3('x')

        DISPLAY "Welcome to FCAI X-O Game. :)"
        DISPLAY "Press 1 if you want to play with computer:"
        READ choice_player

        IF choice_player != 1 THEN
            players[1] = new Player3('o')
        ELSE
            players[1] = new RandomPlayer3('o', 5) // change dimension
            CREATE x_o_game3(new Game3_X_O_Board, players)
            RUN x_o_game3
            EXECUTE system("pause")

        ELSE IF choice == 5 THEN
            DISPLAY "Thank you for using our games ^_^ see you later ^_^"
            BREAK

        ELSE
            DISPLAY "Invalid choice"
            DISPLAY "Good Bye see you later ^_^"
            BREAK
        END IF
    END WHILE
END
```

GitHub:



Commits on Dec 17, 2023

Final Edit Menu
SH-code12 committed 14 minutes ago
Verified 6caf6be

Edit Game3
SH-code12 committed 15 minutes ago
Verified a10ca7e

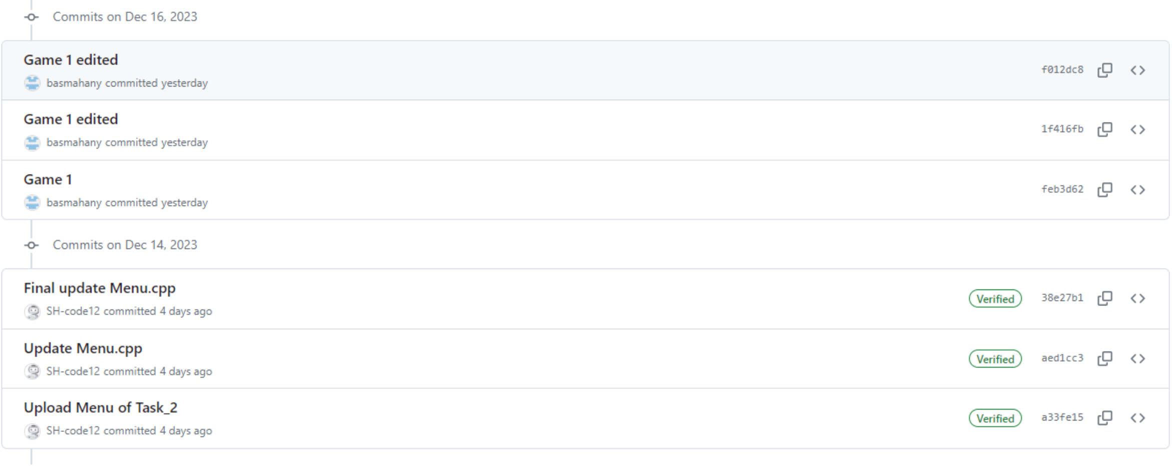
Edit Game1
SH-code12 committed 17 minutes ago
Verified 4ee6785

Edit Game2
SH-code12 committed 17 minutes ago
Verified f5e2c34

Edit Game1
SH-code12 committed 18 minutes ago
Verified 32873fb

Game 2 ···
basmahany committed 3 hours ago
4aff15a

Game 1 header file
basmahany committed 5 hours ago
b01a473



Commits on Dec 16, 2023

Game 1 edited
basmahany committed yesterday
f012dc8

Game 1 edited
basmahany committed yesterday
1f416fb

Game 1
basmahany committed yesterday
feb3d62

Commits on Dec 14, 2023

Final update Menu.cpp
SH-code12 committed 4 days ago
Verified 38e27b1

Update Menu.cpp
SH-code12 committed 4 days ago
Verified aed1cc3

Upload Menu of Task_2
SH-code12 committed 4 days ago
Verified a33fe15