Comprehensive Exercise Report

Team ALBcreators

<<Group Member names and unity IDs

Simeon Flamuraj-230ADB027

Adela Muskollari-230AIB019>>

NOTE: You will replace all placeholders that are given in <<>>

[**Requirements/Analysis**](#_uwgqwd5ezv2w) **2**

[Journal](#_lsityg2iq9m6) 2

[Software Requirements](#_2h0vru1u2mla) 3

[**Black-Box Testing**](#_prhaxdxmf8n8) **4**

[Journal](#_18f11w613jft) 4

[Black-box Test Cases](#_2xn4jzot820y) 5

[**Design**](#_24fdizefyocn) **6**

[Journal](#_esp2ocs9j6bk) 6

[Software Design](#_aifbl1x6rddt) 7

[**Implementation**](#_hya8f3jqkba6) **8**

[Journal](#_acupzfhai7gz) 8

[Implementation Details](#_ojhtwkms2z3b) 9

[**Testing**](#_3qvya3vi836q) **10**

[Journal](#_ckfs4xbl5pyr) 10

[Testing Details](#_bzt1547yxzxi) 11

[**Presentation**](#_hdjvrbf45b1p) **12**

[Preparation](#_xbiquwtmf36n) 12

[**Grading Rubric**](#_u0hfnmdgusmf) **13**

# Requirements/Analysis

Week 2

## Journal

The following prompts are meant to aid your thought process as you complete the requirements/analysis portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

* After reading the client’s brief (possibly incomplete description), write one sentence that describes the project (expected software) and list the already known requirements.
  + <<The client described a development of a game called “connect four” which will require 2 players and two different colored stones to play on a grid board.>>
    - <<At the game the first one who connect four stones in any way in same colors, for example one player red and another yellow, will win the match and gain some points and who ever reaches 100 points will win the game, 1 match is 20 points in case of a tie 10 points for each player >>
* After reading the client’s brief (possibly incomplete description), what questions do you have for the client? Are there any pieces that are unclear? After you have a list of questions, raise your hand and ask the client (your instructor) the questions; make sure to document his/her answers.
  + <<1-Is there any way to get bonus points?

2-Is there a specific number of stones for the players?

3-Is there any chance to lose points?>>

* Does the project cover topics you are unfamiliar with? If so, look up the topics and list your references.
  + <<Fairly familiar-We have played a similar game before and looked it in the references below:
  + 1-https://www.wikihow.com/Play-Connect-4
  + 2-https://instructions.hasbro.com/en-my/instruction/connect-4-game>>
* Describe the users of this software (e.g., small child, high school teacher who is taking attendance).
  + <<Aimed to 6+ audience >>
* Describe how each user would interact with the software
  + <<The user will interact through a web page>>
* What features must the software have? What should the users be able to do?
  + <<First of all, it would be interactive and the player is able to choose where he will put the stones to make his own strategy to win.>>
* Other notes:
  + <<Insert notes>>

## Software Requirements

<<

* Software Requirements:

1. In the game would be 56 stones 28 red ones and 28 yellow ones.
2. The game must have a board with size 6x7.
3. The game mode is only two players.
4. Before every turn the code is checked for a win or draw condition.
5. After the win or draw the points are registered to the right player,(win-10 points, draw-5 points each)
6. The game constantly check if a player has reached maximum points therefor the winner.(30 points in case of a tie which is rare an extra round will start to determine the winner)

* User stories:

1. Like a player, I want to choose where I put my stones.
2. As a player, I want to see how many points I have.
3. As a player, I want to know the maximum points to win.
4. As a player, I want to start a new game after reaching maximum points.
5. As a player, I want to exit the game mid-way.

>>

# Black-Box Testing

Instructions: Week 4

## Journal

***Remember:*** Black box tests should only be based on your requirements and should work independent of design.

The following prompts are meant to aid your thought process as you complete the black box testing portion of this exercise. Please review your list of requirements and respond to each of the prompts below. Feel free to add additional notes.

* What does input for the software look like (e.g., what type of data, how many pieces of data)?
  + << The input would be from the players to choose where they put the stones in the board which is the size 6X7, but the stones will be always at the last free grid from the bottom, till they connect 4 stones with the same color.>>
* What does output for the software look like (e.g., what type of data, how many pieces of data)?
  + << After every turn the new stone will be added to the board represented as matrix in the code meanwhile checking for the win or draw>>
* What equivalence classes can the input be broken into?
  + <<Board:6 collumns X 7 rows
  + Stones:28 stones each player(red and black)
  + Win\_condictions: horizontal, vertical, diagonal
  + Draw\_conditions
  + Menu
  + Number of players:2
  + >>
* What boundary values exist for the input?
  + << fixed board size
  + Number of the players only 2
  + No timer >>
* Are there other cases that must be tested to test all requirements?
  + <<Testing for not correct coordinates
  + Testing for invalid input
  + >>
* Other notes:
  + <<Insert notes>>

## Black-box Test Cases

Use your notes from above to complete the black-box test plan section of the formal documentation by writing black box test cases (other than actual results since no program currently exists). Remember to test each equivalence class, boundary value, and requirement.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Expected Results** | **Actual Results** |
| 1 | Testing the displayBoard | The boar 6X7 will be printed | yes |
| 2 | Testing the Graphical interface(GUI) | The menu will pop up for starting the game | yes |
| 3 | Starting a new game | A new game will start and random player will start probably red | yes |
| 4 | Testing when the turns are switched | When the player 1 will finish his turn the player 2 will be able to put his stone | yes |
| 5 | Testing the win for each player | When the player 1 or the player 2 will connect 4 he will win points and the next round would start | yes |
| 6 | Testing the draw | When the players will ran out of stones or moves the fame will end with a tie and the next round will start | N/A |
| 7 | Testing the points for the win conditions for each player | When the player wins the score of that round will be shown and registered | yes |
| 8 | Testing the points for the draws condition | When the game ends with draw a same amount of the points is added to both players | N/A |
| 9 | Testing when player 1 or player 2 have similar points at maximum points an extra round will start | When the players at the maximum stores ends up with similar points the extra round will decide who is the winner | yes |
| 10 | Testing when player 1 or player 2 reach the maximum points | When the player reach the maximum points he will win the game and the menu will pop up for starting a new game | yes |

# Design

Instructions: Week 6

## Journal

***Remember:*** You still will not be writing code at this point in the process.

The following prompts are meant to aid your thought process as you complete the design portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

* List the nouns from your requirements/analysis documentation.
  + Player
  + Main
  + Play\_AI
  + Game
  + Stone
  + Size
* Which nouns potentially may represent a class in your design?
  + play
  + play\_AI
  + Manin
* Which nouns potentially may represent attributes/fields in your design? Also list the class each attribute/field would be a part of.
  + button
  + size
  + board
  + stones
  + turn
* Now that you have a list of possible classes, consider different design options (***lists of classes and attributes***) along with the pros and cons of each. We often do not come up with the best design on our first attempt. Also consider whether any needed classes are missing. These two design options should not be GUI vs. non-GUI; instead you need to include the classes and attributes for each design. Reminder: Each design must include at least two classes that define object types.
  + <<List at least two design options with pros and cons of each>>
  + Player
  + Game

Pros:

* Simple and straightforward with only two classes
* Matches a real game that is created by the game itself and the player
* Can be easy to implement for a simple game

Cons:

* May not handle more complex game algorithms
* May not include all classes and attributes to make the game more satisfactory for players
* Which design do you plan to use? Explain why you have chosen this design.  
  The system is relatively small and simple, and a more complex design would be overkill. The system is a game so these two classes are the most relevant ones. Most players would like a simple game that it can also be played by different group ages.
* List the verbs from your requirements/analysis documentation.
  + For the noun Us the verb is designing (since we are designing the system)
  + For the noun User the verbs are playing, views the points of the game
  + For the noun Game verb is storing the player’s game information
* Which verbs potentially may represent a method in your design? Also list the class each method would be part of.
  + startGame(): This method is part of Game class and would allow a new game to start
  + finishGame(): This method is also part of Game class and would allow to finish the game
  + winning\_move()- check the board and returns the winner if there is a winner
  + reset\_board()- after a winning the board reset to the start points.
* Other notes:
  + <<Insert notes>>

## Software Design

<<Use your notes from above to complete this section of the formal documentation by planning the classes, methods, and fields that will used in the software. Your design should include UML class diagrams along with method headers. ***Prior to starting the formal documentation, you should show your answers to the above prompts to your instructor.****>>*

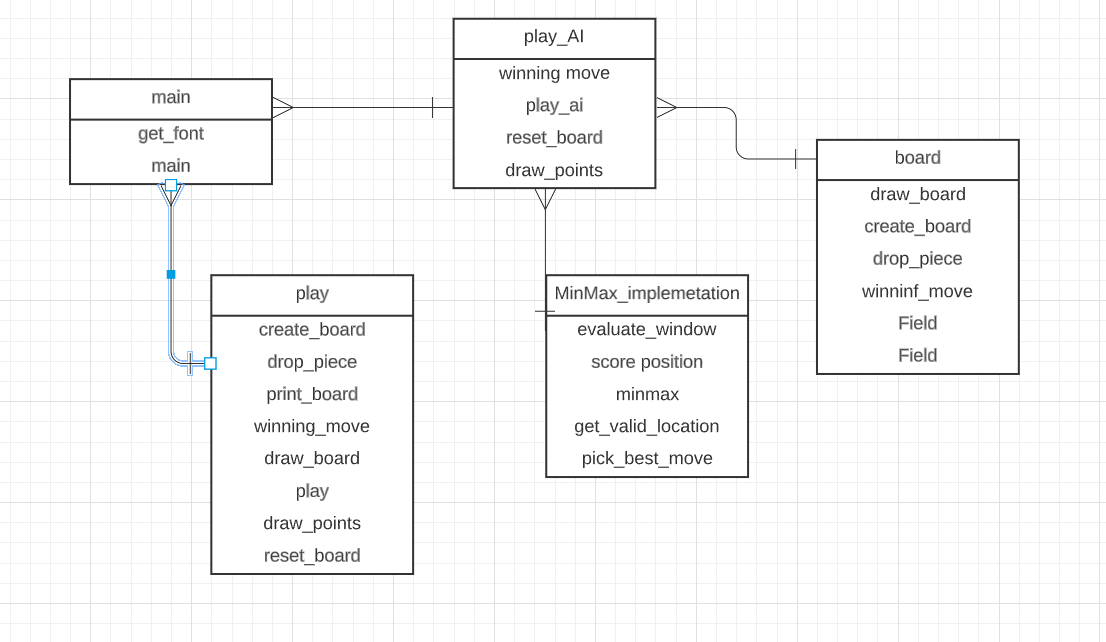
mentioned for the design there will be two classes Player and Game but also additional classes might be implemented later. This design will be simple and straightforward, every age group will be able to play it.  
The moment the play enters the game they can start playing by pressing play, if they wish to end the game sooner, they can press finish game. There will also be the option that they can view the points they have received for each game they have won.

|  |
| --- |
| User |
| * Name * User Id |

|  |
| --- |
| Player |
| * lookPoints() * playsGame() |

|  |
| --- |
| Game |
| * startGame() * finishGame() |

The actual UML design is:



# Implementation

Instructions: Week 8

## Journal

The following prompts are meant to aid your thought process as you complete the implementation portion of this exercise. Please respond to each of the prompt below and feel free to add additional notes.

* What programming concepts from the course will you need to implement your design? Briefly explain how each will be used during implementation.
  + <<From the notes I know the main points to translate to the code, but not all classes or what is needed is found out in the design process but while coding it. >>
* Other notes:
  + <<Insert notes>>

## Implementation Details

* First file: main.py
* from button import Button  
  import pygame  
  import os  
  import sys  
  from play import play  
  from play\_AI import play\_ai  
    
  WIDTH = 800  
  HIGHT = 700  
  BLACK = (0, 0, 0)  
  BUTTON\_C=(215, 252, 212)  
  TITLE\_C=(182, 143, 64)  
  pygame.init() # initialize Pygame modules, including the font module  
    
  WIN = pygame.display.set\_mode((WIDTH, HIGHT))  
  pygame.display.set\_caption("Connect-4")  
  print("window created")  
  background = pygame.transform.scale(pygame.image.load(os.path.join('ASSET', 'download.jpg')), (WIDTH, HIGHT))  
  menu\_font = pygame.font.SysFont('comicsans', 100)  
    
  def get\_font(size):  
   return pygame.font.SysFont('comicsans', size)  
    
  def main():  
   while True:  
   WIN.blit(background, (0, 0))  
    
   TITLE = get\_font(60).render("Welcome to Connect-4", 1, TITLE\_C)  
   TITLE\_RECT= TITLE.get\_rect(center=(400, 150))  
   WIN.blit(TITLE, TITLE\_RECT)  
   UNA = get\_font(15).render("by Simeon Flamuraj and Adela Muskollari", 1, (231, 231, 231))  
   UNA\_RECT = UNA.get\_rect(center=(400, 200))  
   WIN.blit(UNA, UNA\_RECT)  
    
   MENU\_MOUSE\_POS = pygame.mouse.get\_pos()  
   print("loop is running")  
    
   PLAY\_BUTTON = Button(image=None, pos=(400, 380),  
   text\_input="PLAY", font=get\_font(40), base\_color=BUTTON\_C, hovering\_color=(200, 0, 200))  
   PLAY\_AI\_BUTTON = Button(image=None, pos=(400, 450),  
   text\_input="PLAY\_AI", font=get\_font(40), base\_color=BUTTON\_C, hovering\_color=(200, 0, 200))  
    
   for button in [PLAY\_BUTTON,PLAY\_AI\_BUTTON]:  
   button.changeColor(MENU\_MOUSE\_POS)  
   button.update(WIN)  
    
   for event in pygame.event.get():  
   if event.type == pygame.QUIT:  
   print("loop finished")  
   sys.exit()  
   if event.type == pygame.MOUSEBUTTONDOWN:  
   if PLAY\_BUTTON.checkForInput(MENU\_MOUSE\_POS):  
   play()  
   if PLAY\_AI\_BUTTON.checkForInput(MENU\_MOUSE\_POS):  
   play\_ai()  
    
    
    
   pygame.display.update()  
    
    
    
    
  if \_\_name\_\_ == "\_\_main\_\_":  
   main()
* board.py
* import numpy as np  
  import pygame  
    
    
  BLUE = (0, 0, 255)  
  BLACK = (0, 0, 0)  
  RED = (255, 0, 0)  
  YELLOW = (255, 255, 0)  
    
  ROW\_COUNT = 6  
  COLUMN\_COUNT = 7  
    
    
  def create\_board():  
   board = np.zeros((ROW\_COUNT, COLUMN\_COUNT))  
   return board  
    
    
  def drop\_piece(board, row, col, piece):  
   board[row][col] = piece  
    
    
  def is\_valid\_location(board, col):  
   return board[ROW\_COUNT - 1][col] == 0  
    
    
  def get\_next\_open\_row(board, col):  
   for r in range(ROW\_COUNT):  
   if board[r][col] == 0:  
   return r  
    
    
  def print\_board(board):  
   print(np.flip(board, 0))  
    
    
  def winning\_move(board, piece):  
   # Check horizontal locations for win  
   for c in range(COLUMN\_COUNT - 3):  
   for r in range(ROW\_COUNT):  
   if board[r][c] == piece and board[r][c + 1] == piece and board[r][c + 2] == piece and board[r][  
   c + 3] == piece:  
   return True  
    
   # Check vertical locations for win  
   for c in range(COLUMN\_COUNT):  
   for r in range(ROW\_COUNT - 3):  
   if board[r][c] == piece and board[r + 1][c] == piece and board[r + 2][c] == piece and board[r + 3][  
   c] == piece:  
   return True  
    
   # Check positively sloped diaganols  
   for c in range(COLUMN\_COUNT - 3):  
   for r in range(ROW\_COUNT - 3):  
   if board[r][c] == piece and board[r + 1][c + 1] == piece and board[r + 2][c + 2] == piece and board[r + 3][  
   c + 3] == piece:  
   return True  
    
   # Check negatively sloped diaganols  
   for c in range(COLUMN\_COUNT - 3):  
   for r in range(3, ROW\_COUNT):  
   if board[r][c] == piece and board[r - 1][c + 1] == piece and board[r - 2][c + 2] == piece and board[r - 3][  
   c + 3] == piece:  
   return True  
    
    
  def draw\_board(board):  
   for c in range(COLUMN\_COUNT):  
   for r in range(ROW\_COUNT):  
   pygame.draw.rect(screen, BLUE, (c \* SQUARESIZE, r \* SQUARESIZE + SQUARESIZE, SQUARESIZE, SQUARESIZE))  
   pygame.draw.circle(screen, BLACK, (  
   int(c \* SQUARESIZE + SQUARESIZE / 2), int(r \* SQUARESIZE + SQUARESIZE + SQUARESIZE / 2)), RADIUS)  
    
   for c in range(COLUMN\_COUNT):  
   for r in range(ROW\_COUNT):  
   if board[r][c] == 1:  
   pygame.draw.circle(screen, RED, (  
   int(c \* SQUARESIZE + SQUARESIZE / 2), height - int(r \* SQUARESIZE + SQUARESIZE / 2)), RADIUS)  
   elif board[r][c] == 2:  
   pygame.draw.circle(screen, YELLOW, (  
   int(c \* SQUARESIZE + SQUARESIZE / 2), height - int(r \* SQUARESIZE + SQUARESIZE / 2)), RADIUS)  
   pygame.display.update()  
    
    
  board = create\_board()  
  print\_board(board)  
  game\_over = False  
  turn = 0  
    
  # initalize pygame  
  pygame.init()  
    
  # define our screen size  
  SQUARESIZE = 100  
    
  # define width and height of board  
  width = COLUMN\_COUNT \* SQUARESIZE  
  height = (ROW\_COUNT + 1) \* SQUARESIZE  
    
  size = (width, height)  
    
  RADIUS = int(SQUARESIZE / 2 - 5)  
    
  screen = pygame.display.set\_mode(size)  
  # Calling function draw\_board again  
  draw\_board(board)  
  pygame.display.update()  
    
  myfont = pygame.font.SysFont("monospace", 75)  
    
    
    
    
  BLUE = (0,0,255)  
  BLACK = (0,0,0)  
  RED = (255,0,0)  
  YELLOW = (255,255,0)  
    
  ROW\_COUNT = 6  
  COLUMN\_COUNT = 7  
    
  def create\_board():  
   board = np.zeros((ROW\_COUNT,COLUMN\_COUNT))  
   return board  
    
  def drop\_piece(board, row, col, piece):  
   board[row][col] = piece  
    
  def is\_valid\_location(board, col):  
   return board[ROW\_COUNT-1][col] == 0  
    
  def get\_next\_open\_row(board, col):  
   for r in range(ROW\_COUNT):  
   if board[r][col] == 0:  
   return r  
    
  def print\_board(board):  
   print(np.flip(board, 0))  
    
  def winning\_move(board, piece):  
   # Check horizontal locations for win  
   for c in range(COLUMN\_COUNT-3):  
   for r in range(ROW\_COUNT):  
   if board[r][c] == piece and board[r][c+1] == piece and board[r][c+2] == piece and board[r][c+3] == piece:  
   return True  
    
   # Check vertical locations for win  
   for c in range(COLUMN\_COUNT):  
   for r in range(ROW\_COUNT-3):  
   if board[r][c] == piece and board[r+1][c] == piece and board[r+2][c] == piece and board[r+3][c] == piece:  
   return True  
    
   # Check positively sloped diaganols  
   for c in range(COLUMN\_COUNT-3):  
   for r in range(ROW\_COUNT-3):  
   if board[r][c] == piece and board[r+1][c+1] == piece and board[r+2][c+2] == piece and board[r+3][c+3] == piece:  
   return True  
    
   # Check negatively sloped diaganols  
   for c in range(COLUMN\_COUNT-3):  
   for r in range(3, ROW\_COUNT):  
   if board[r][c] == piece and board[r-1][c+1] == piece and board[r-2][c+2] == piece and board[r-3][c+3] == piece:  
   return True  
    
  def draw\_board(board):  
   for c in range(COLUMN\_COUNT):  
   for r in range(ROW\_COUNT):  
   pygame.draw.rect(screen, BLUE, (c\*SQUARESIZE, r\*SQUARESIZE+SQUARESIZE, SQUARESIZE, SQUARESIZE))  
   pygame.draw.circle(screen, BLACK, (int(c\*SQUARESIZE+SQUARESIZE/2), int(r\*SQUARESIZE+SQUARESIZE+SQUARESIZE/2)), RADIUS)  
    
   for c in range(COLUMN\_COUNT):  
   for r in range(ROW\_COUNT):  
   if board[r][c] == 1:  
   pygame.draw.circle(screen, RED, (int(c\*SQUARESIZE+SQUARESIZE/2), height-int(r\*SQUARESIZE+SQUARESIZE/2)), RADIUS)  
   elif board[r][c] == 2:  
   pygame.draw.circle(screen, YELLOW, (int(c\*SQUARESIZE+SQUARESIZE/2), height-int(r\*SQUARESIZE+SQUARESIZE/2)), RADIUS)  
   pygame.display.update()
* Play.py
* from board import \*  
  import numpy as np  
  import pygame  
  import math  
    
  board = create\_board()  
  print\_board(board)  
  game\_over = False  
  turn = 0  
    
  # initalize pygame  
  pygame.init()  
    
  # define our screen size  
  SQUARESIZE = 100  
    
  # define width and height of board  
  width = COLUMN\_COUNT \* SQUARESIZE  
  height = (ROW\_COUNT + 1) \* SQUARESIZE  
    
  size = (width, height)  
    
  RADIUS = int(SQUARESIZE / 2 - 5)  
    
  screen = pygame.display.set\_mode(size)  
  # Calling function draw\_board again  
  draw\_board(board)  
  pygame.display.update()  
    
  myfont = pygame.font.SysFont("monospace", 75)  
    
  def get\_font(size):  
   return pygame.font.SysFont('comicsans', size)  
    
    
    
    
  BLUE = (0,0,255)  
  BLACK = (0,0,0)  
  RED = (255,0,0)  
  YELLOW = (255,255,0)  
    
  ROW\_COUNT = 6  
  COLUMN\_COUNT = 7  
    
  def create\_board():  
   board = np.zeros((ROW\_COUNT,COLUMN\_COUNT))  
   return board  
    
  def drop\_piece(board, row, col, piece):  
   board[row][col] = piece  
    
  def is\_valid\_location(board, col):  
   return board[ROW\_COUNT-1][col] == 0  
    
  def get\_next\_open\_row(board, col):  
   for r in range(ROW\_COUNT):  
   if board[r][col] == 0:  
   return r  
    
  def print\_board(board):  
   print(np.flip(board, 0))  
    
  def winning\_move(board, piece):  
   # Check horizontal locations for win  
   for c in range(COLUMN\_COUNT-3):  
   for r in range(ROW\_COUNT):  
   if board[r][c] == piece and board[r][c+1] == piece and board[r][c+2] == piece and board[r][c+3] == piece:  
   return True  
    
   # Check vertical locations for win  
   for c in range(COLUMN\_COUNT):  
   for r in range(ROW\_COUNT-3):  
   if board[r][c] == piece and board[r+1][c] == piece and board[r+2][c] == piece and board[r+3][c] == piece:  
   return True  
    
   # Check positively sloped diaganols  
   for c in range(COLUMN\_COUNT-3):  
   for r in range(ROW\_COUNT-3):  
   if board[r][c] == piece and board[r+1][c+1] == piece and board[r+2][c+2] == piece and board[r+3][c+3] == piece:  
   return True  
    
   # Check negatively sloped diaganols  
   for c in range(COLUMN\_COUNT-3):  
   for r in range(3, ROW\_COUNT):  
   if board[r][c] == piece and board[r-1][c+1] == piece and board[r-2][c+2] == piece and board[r-3][c+3] == piece:  
   return True  
    
  def draw\_board(board):  
   for c in range(COLUMN\_COUNT):  
   for r in range(ROW\_COUNT):  
   pygame.draw.rect(screen, BLUE, (c\*SQUARESIZE, r\*SQUARESIZE+SQUARESIZE, SQUARESIZE, SQUARESIZE))  
   pygame.draw.circle(screen, BLACK, (int(c\*SQUARESIZE+SQUARESIZE/2), int(r\*SQUARESIZE+SQUARESIZE+SQUARESIZE/2)), RADIUS)  
    
   for c in range(COLUMN\_COUNT):  
   for r in range(ROW\_COUNT):  
   if board[r][c] == 1:  
   pygame.draw.circle(screen, RED, (int(c\*SQUARESIZE+SQUARESIZE/2), height-int(r\*SQUARESIZE+SQUARESIZE/2)), RADIUS)  
   elif board[r][c] == 2:  
   pygame.draw.circle(screen, YELLOW, (int(c\*SQUARESIZE+SQUARESIZE/2), height-int(r\*SQUARESIZE+SQUARESIZE/2)), RADIUS)  
   pygame.display.update()  
    
    
  def play():  
   player1=0  
   player2=0  
   board = create\_board()  
   print\_board(board)  
   game\_over = False  
   turn = 0  
    
   #initalize pygame  
   pygame.init()  
    
   #define our screen size  
   SQUARESIZE = 100  
    
   #define width and height of board  
   width = COLUMN\_COUNT \* SQUARESIZE  
   height = (ROW\_COUNT+1) \* SQUARESIZE  
    
   size = (width, height)  
    
   RADIUS = int(SQUARESIZE/2 - 5)  
    
   screen = pygame.display.set\_mode(size)  
   #Calling function draw\_board again  
   draw\_board(board)  
   pygame.display.update()  
    
   myfont = pygame.font.SysFont("monospace", 75)  
    
    
    
   while not game\_over:  
    
   for event in pygame.event.get():  
   if event.type == pygame.QUIT:  
   return  
    
   if event.type == pygame.MOUSEMOTION:  
    
   pygame.draw.rect(screen, BLACK, (0,0, width, SQUARESIZE))  
   posx = event.pos[0]  
   draw\_points(1, player1, (0, 0),RED)  
   draw\_points(2, player2, (600, 0),YELLOW)  
   if turn == 0:  
   pygame.draw.circle(screen, RED, (posx, int(SQUARESIZE/2)), RADIUS)  
   else:  
   pygame.draw.circle(screen, YELLOW, (posx, int(SQUARESIZE/2)), RADIUS)  
   pygame.display.update()  
    
   if event.type == pygame.MOUSEBUTTONDOWN:  
   pygame.draw.rect(screen, BLACK, (0,0, width, SQUARESIZE))  
   print(event.pos)  
   # Ask for Player 1 Input  
   if turn == 0:  
   posx = event.pos[0]  
   col = int(math.floor(posx/SQUARESIZE))  
    
   if is\_valid\_location(board, col):  
   row = get\_next\_open\_row(board, col)  
   drop\_piece(board, row, col, 1)  
    
   if winning\_move(board, 1)==True:  
   player1+=10  
   if player1==20:  
   player1 = 0  
   player2 = 0  
   label = myfont.render("Player 1 wins!!", 1, RED)  
   screen.blit(label, (40,10))  
   pygame.display.update()  
   pygame.time.wait(3000)  
   return  
   elif board\_full(board):  
    
   player1 += 5  
   player2 += 5  
    
   reset\_board(board)  
   else:  
   reset\_board(board)  
    
    
    
   # # Ask for Player 2 Input  
   else:  
   posx = event.pos[0]  
   col = int(math.floor(posx/SQUARESIZE))  
    
   if is\_valid\_location(board, col):  
   row = get\_next\_open\_row(board, col)  
   drop\_piece(board, row, col, 2)  
    
   if winning\_move(board, 2):  
   player2+=10  
   if player2 ==20:  
   player1=0  
   player2=0  
   label = myfont.render("Player 2 wins!!", 1, YELLOW)  
   screen.blit(label, (40,10))  
   pygame.display.update()  
   pygame.time.wait(3000)  
   return  
   elif board\_full(board):  
    
   player1 += 5  
   player2 += 5  
    
   reset\_board(board)  
   else:  
   reset\_board(board)  
    
    
    
    
   print\_board(board)  
   draw\_board(board)  
    
    
    
    
   turn += 1  
   turn = turn % 2  
   pygame.display.update()  
    
    
    
    
    
    
    
  def draw\_points(n, points,pos,color):  
   PROV = get\_font(20).render(f'player{n}={points}', 1, color)  
   screen.blit(PROV, pos)  
    
    
    
    
  def reset\_board(board):  
   # Set all cells to 0  
   for r in range(ROW\_COUNT):  
   for c in range(COLUMN\_COUNT):  
   board[r][c] = 0  
    
    
    
    
    
  def board\_full(board):  
   for row in board:  
   if 0 in row:  
   return False  
   return True
* play\_ai.py
* from MinMax\_implemetation import \*  
    
  WIDTH = 800  
  HIGHT = 700  
  BLACK = (0, 0, 0)  
    
  def get\_font(size):  
   return pygame.font.SysFont('comicsans', size)  
    
  ROW\_COUNT = 6  
  COLUMN\_COUNT = 7  
    
  pygame.init() # initialize Pygame modules, including the font module  
    
  screen = pygame.display.set\_mode((WIDTH, HIGHT))  
  pygame.display.set\_caption("Connect-4")  
    
  def winning\_move(board, piece):  
   # Check horizontal locations for win  
   for c in range(COLUMN\_COUNT-3):  
   for r in range(ROW\_COUNT):  
   if board[r][c] == piece and board[r][c+1] == piece and board[r][c+2] == piece and board[r][c+3] == piece:  
   return True  
    
   # Check vertical locations for win  
   for c in range(COLUMN\_COUNT):  
   for r in range(ROW\_COUNT-3):  
   if board[r][c] == piece and board[r+1][c] == piece and board[r+2][c] == piece and board[r+3][c] == piece:  
   return True  
    
   # Check positively sloped diaganols  
   for c in range(COLUMN\_COUNT-3):  
   for r in range(ROW\_COUNT-3):  
   if board[r][c] == piece and board[r+1][c+1] == piece and board[r+2][c+2] == piece and board[r+3][c+3] == piece:  
   return True  
    
   # Check negatively sloped diaganols  
   for c in range(COLUMN\_COUNT-3):  
   for r in range(3, ROW\_COUNT):  
   if board[r][c] == piece and board[r-1][c+1] == piece and board[r-2][c+2] == piece and board[r-3][c+3] == piece:  
   return True  
    
    
    
    
  def play\_ai():  
   board = create\_board()  
   print\_board(board)  
   game\_over = False  
   player1=0  
   player2=0  
    
   pygame.init()  
    
   SQUARESIZE = 100  
    
   width = COLUMN\_COUNT \* SQUARESIZE  
   height = (ROW\_COUNT + 1) \* SQUARESIZE  
    
   size = (width, height)  
    
   RADIUS = int(SQUARESIZE / 2 - 5)  
    
   screen = pygame.display.set\_mode(size)  
   draw\_board(board)  
   pygame.display.update()  
    
   myfont = pygame.font.SysFont("monospace", 75)  
    
   turn = random.randint(PLAYER, AI)  
    
   while not game\_over:  
    
   for event in pygame.event.get():  
   if event.type == pygame.QUIT:  
   return  
    
   if event.type == pygame.MOUSEMOTION:  
   pygame.draw.rect(screen, BLACK, (0, 0, width, SQUARESIZE))  
   posx = event.pos[0]  
   draw\_points(1, player1, (0, 0), RED)  
   draw\_points(2, player2, (600, 0), YELLOW)  
   if turn == PLAYER:  
   pygame.draw.circle(screen, RED, (posx, int(SQUARESIZE / 2)), RADIUS)  
    
   pygame.display.update()  
    
   if event.type == pygame.MOUSEBUTTONDOWN:  
   pygame.draw.rect(screen, BLACK, (0, 0, width, SQUARESIZE))  
   # print(event.pos)  
   # Ask for Player 1 Input  
   if turn == PLAYER:  
   posx = event.pos[0]  
   col = int(math.floor(posx / SQUARESIZE))  
    
   if is\_valid\_location(board, col):  
   row = get\_next\_open\_row(board, col)  
   drop\_piece(board, row, col, PLAYER\_PIECE)  
    
   if winning\_move(board, PLAYER\_PIECE):  
    
   player1 += 10  
   if player1 == 20:  
   player1 = 0  
   player2 = 0  
   label = myfont.render("Player 1 wins!!", 1, RED)  
   screen.blit(label, (40, 10))  
   pygame.display.update()  
   pygame.time.wait(3000)  
   return  
    
   else:  
   reset\_board(board)  
    
   turn += 1  
   turn = turn % 2  
    
   print\_board(board)  
   draw\_board(board)  
    
   # # Ask for Player 2 Input  
   if turn == AI and not game\_over:  
    
   # col = random.randint(0, COLUMN\_COUNT-1)  
   # col = pick\_best\_move(board, AI\_PIECE)  
   pygame.time.wait(300)  
   col, minimax\_score = minimax(board, 3, -math.inf, math.inf, True)  
    
   if is\_valid\_location(board, col):  
   # pygame.time.wait(500)  
   row = get\_next\_open\_row(board, col)  
   drop\_piece(board, row, col, AI\_PIECE)  
    
   if winning\_move(board, AI\_PIECE):  
    
   player2 += 10  
   if player2 == 20:  
   player1 = 0  
   player2 = 0  
   label = myfont.render("Player 2 wins!!", 1, YELLOW)  
   screen.blit(label, (40, 10))  
   pygame.display.update()  
   pygame.time.wait(3000)  
   return  
   else:  
   reset\_board(board)  
    
   print\_board(board)  
   draw\_board(board)  
    
   turn += 1  
   turn = turn % 2  
    
    
  def reset\_board(board):  
   # Set all cells to 0  
   for r in range(ROW\_COUNT):  
   for c in range(COLUMN\_COUNT):  
   board[r][c] = 0  
    
    
  def draw\_points(n, points,pos,color):  
   PROV = get\_font(20).render(f'player{n}={points}', 1, color)  
   screen.blit(PROV, pos)
* Minmax\_implimitation.py
* import math  
  import random  
    
  from board import \*  
    
  ROW\_COUNT = 6  
  COLUMN\_COUNT = 7  
    
  PLAYER = 0  
  AI = 1  
    
  EMPTY = 0  
  PLAYER\_PIECE = 1  
  AI\_PIECE = 2  
    
  WINDOW\_LENGTH = 4  
    
  def evaluate\_window(window, piece):  
   score = 0  
   opp\_piece = PLAYER\_PIECE  
   if piece == PLAYER\_PIECE:  
   opp\_piece = AI\_PIECE  
    
   if window.count(piece) == 4:  
   score += 100  
   elif window.count(piece) == 3 and window.count(EMPTY) == 1:  
   score += 5  
   elif window.count(piece) == 2 and window.count(EMPTY) == 2:  
   score += 2  
    
   if window.count(opp\_piece) == 3 and window.count(EMPTY) == 1:  
   score -= 4  
    
   return score  
    
    
  def score\_position(board, piece):  
   score = 0  
    
   ## Score center column  
   center\_array = [int(i) for i in list(board[:, COLUMN\_COUNT // 2])]  
   center\_count = center\_array.count(piece)  
   score += center\_count \* 3  
    
   ## Score Horizontal  
   for r in range(ROW\_COUNT):  
   row\_array = [int(i) for i in list(board[r, :])]  
   for c in range(COLUMN\_COUNT - 3):  
   window = row\_array[c:c + WINDOW\_LENGTH]  
   score += evaluate\_window(window, piece)  
    
   ## Score Vertical  
   for c in range(COLUMN\_COUNT):  
   col\_array = [int(i) for i in list(board[:, c])]  
   for r in range(ROW\_COUNT - 3):  
   window = col\_array[r:r + WINDOW\_LENGTH]  
   score += evaluate\_window(window, piece)  
    
   ## Score posiive sloped diagonal  
   for r in range(ROW\_COUNT - 3):  
   for c in range(COLUMN\_COUNT - 3):  
   window = [board[r + i][c + i] for i in range(WINDOW\_LENGTH)]  
   score += evaluate\_window(window, piece)  
    
   for r in range(ROW\_COUNT - 3):  
   for c in range(COLUMN\_COUNT - 3):  
   window = [board[r + 3 - i][c + i] for i in range(WINDOW\_LENGTH)]  
   score += evaluate\_window(window, piece)  
    
   return score  
    
    
  def is\_terminal\_node(board):  
   return winning\_move(board, PLAYER\_PIECE) or winning\_move(board, AI\_PIECE) or len(get\_valid\_locations(board)) == 0  
    
    
    
    
    
  def minimax(board, depth, alpha, beta, maximizingPlayer):  
    
   valid\_locations = get\_valid\_locations(board)  
   is\_terminal = is\_terminal\_node(board)  
   if depth == 0 or is\_terminal:  
   if is\_terminal:  
   if winning\_move(board, AI\_PIECE):  
   return (None, 100000000000000)  
   elif winning\_move(board, PLAYER\_PIECE):  
   return (None, -10000000000000)  
   else: # Game is over, no more valid moves  
   return (None, 0)  
   else: # Depth is zero  
   return (None, score\_position(board, AI\_PIECE))  
   if maximizingPlayer:  
   value = -math.inf  
   column = random.choice(valid\_locations)  
   for col in valid\_locations:  
   row = get\_next\_open\_row(board, col)  
   b\_copy = board.copy()  
   drop\_piece(b\_copy, row, col, AI\_PIECE)  
   new\_score = minimax(b\_copy, depth - 1, alpha, beta, False)[1]  
   if new\_score > value:  
   value = new\_score  
   column = col  
   alpha = max(alpha, value)  
   if alpha >= beta:  
   break  
   return column, value  
    
   else: # Minimizing player  
   value = math.inf  
   column = random.choice(valid\_locations)  
   for col in valid\_locations:  
   row = get\_next\_open\_row(board, col)  
   b\_copy = board.copy()  
   drop\_piece(b\_copy, row, col, PLAYER\_PIECE)  
   new\_score = minimax(b\_copy, depth - 1, alpha, beta, True)[1]  
   if new\_score < value:  
   value = new\_score  
   column = col  
   beta = min(beta, value)  
   if alpha >= beta:  
   break  
   return column, value  
    
    
  def get\_valid\_locations(board):  
   valid\_locations = []  
   for col in range(COLUMN\_COUNT):  
   if is\_valid\_location(board, col):  
   valid\_locations.append(col)  
   return valid\_locations  
    
    
  def pick\_best\_move(board, piece):  
   valid\_locations = get\_valid\_locations(board)  
   best\_score = -10000  
   best\_col = random.choice(valid\_locations)  
   for col in valid\_locations:  
   row = get\_next\_open\_row(board, col)  
   temp\_board = board.copy()  
   drop\_piece(temp\_board, row, col, piece)  
   score = score\_position(temp\_board, piece)  
   if score > best\_score:  
   best\_score = score  
   best\_col = col  
    
   return best\_col
* button.py

class Button():  
 *"""  
 This class is used to create a button with a given image, position, text, font, base color and hovering color.  
 It also has methods to update the button, check for input and change the color of the button.  
 """* def \_\_init\_\_(self, image, pos, text\_input, font, base\_color, hovering\_color):  
 *"""  
 This is the constructor for the Button class. It takes in the following parameters:  
 image: The image to be used for the button.  
 pos: The position of the button.  
 text\_input: The text to be displayed on the button.  
 font: The font to be used for the text.  
 base\_color: The base color of the button.  
 hovering\_color: The color of the button when it is being hovered over.  
 """* self.image = image  
 self.x\_pos = pos[0]  
 self.y\_pos = pos[1]  
 self.font = font  
 self.base\_color, self.hovering\_color = base\_color, hovering\_color  
 self.text\_input = text\_input  
 self.text = self.font.render(self.text\_input, True, self.base\_color)  
 if self.image is None:  
 self.image = self.text  
 self.rect = self.image.get\_rect(center=(self.x\_pos, self.y\_pos))  
 self.text\_rect = self.text.get\_rect(center=(self.x\_pos, self.y\_pos))  
  
 def update(self, screen):  
 *"""  
 This method updates the button on the given screen.  
 """* if self.image is not None:  
 screen.blit(self.image, self.rect)  
 screen.blit(self.text, self.text\_rect)  
  
 def checkForInput(self, position):  
 *"""  
 This method checks if the given position is within the bounds of the button.  
 It returns True if it is, False otherwise.  
 """* if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):  
 return True  
 return False  
  
 def changeColor(self, position):  
 *"""  
 This method changes the color of the button based on the given position.  
 If the position is within the bounds of the button, the color is changed to the hovering color.  
 Otherwise, the color is changed to the base color.  
 """* if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):  
 self.text = self.font.render(self.text\_input, True, self.hovering\_color)  
 else:  
 self.text = self.font.render(self.text\_input, True, self.base\_color)

# Testing

Instructions: Week 10

## Journal

The following prompts are meant to aid your thought process as you complete the testing portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

* Have you changed any requirements since you completed the black box test plan? If so, list changes below and update your black-box test plan appropriately.
  + <<No we haven’t>>
* List the classes of your implementation. For each class, list equivalence classes, boundary values, and paths through code that you should test.

Play():

setUp()- for the pygame.init()

tearDown() – for quitting the game

test\_create\_board() – for the board

test\_drop\_piece() – test the pieces

test\_get\_next\_open\_row()

test\_winning\_move()- test the winning conditions

test\_reset\_board()- test if the board resets

## 

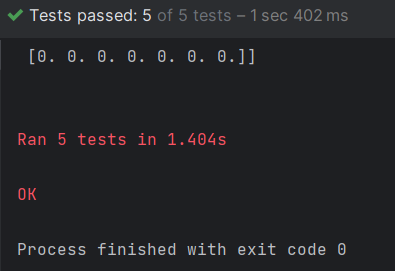
## 

## Testing Details

The testing code is below:  
input:

import unittest  
from unittest.mock import patch  
import pygame  
from board import \*  
from play import reset\_board  
  
  
class TestConnect4(unittest.TestCase):  
 def setUp(self):  
 pygame.init()  
  
 def tearDown(self):  
 pygame.quit()  
  
 def test\_create\_board(self):  
 board = create\_board()  
 self.assertEqual(board.shape, (ROW\_COUNT, COLUMN\_COUNT))  
 self.assertTrue(np.all(board == 0))  
  
 def test\_drop\_piece(self):  
 board = create\_board()  
 drop\_piece(board, 0, 0, 1)  
 self.assertEqual(board[0][0], 1)  
  
  
 def test\_get\_next\_open\_row(self):  
 board = create\_board()  
 drop\_piece(board, 0, 0, 1)  
 drop\_piece(board, 1, 0, 1)  
 drop\_piece(board, 2, 0, 2)  
 self.assertEqual(get\_next\_open\_row(board, 0), 3)  
 self.assertEqual(get\_next\_open\_row(board, 1), 0)  
  
 def test\_winning\_move(self):  
 board = create\_board()  
 drop\_piece(board, 0, 0, 1)  
 drop\_piece(board, 0, 1, 1)  
 drop\_piece(board, 0, 2, 1)  
 drop\_piece(board, 0, 3, 1)  
 self.assertTrue(winning\_move(board, 1))  
  
 def test\_reset\_board(self):  
 board = create\_board()  
 drop\_piece(board, 0, 0, 1)  
 drop\_piece(board, 1, 0, 1)  
 drop\_piece(board, 2, 0, 2)  
 reset\_board(board)  
 self.assertTrue(np.all(board == 0))  
  
  
  
  
@patch('builtins.input', return\_value='1') # Mocks the input function and returns '1'  
def test\_user\_input(self, input\_mock):  
 # Your test code that involves user input  
  
 if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

Output:



# Presentation

Instructions:Week 12

## Preparation

The following prompts are meant to aid your thought process as you complete the presentation portion of this exercise. It is recommended that you examine the previous sections of the journal and your reflections as you work on the presentation as it is likely that you have already answered some of the following prompts elsewhere. Please respond to each of the prompts below and feel free to add additional notes.

* Give a brief description of your final project
  + <<The project started by choosing which game will we code, we chose connect 4, but not the original game with small difference. The difference of our game is that our version is with points and the player has to get specific points to win. The game allows you to play with two players or with the computer where who starts the game is randomly chosen. For the computer player we chosen the Min-Max algorithm with Alfa-Beta pruning.>>
* Describe your requirement assumptions/additions.
  + The game should have a two player mode
  + As a player I want to quit mid-game
  + The points should be clear
  + At the start of the game there should be a menu where I chose to player a two player mode or with the computer.
* Describe your design options and decision. How did you weigh the pros and cons of the different designs to make your decision?
  + From the previous design I knew what were the main classes and functions that I needed to translate to code so the pros and cons of the design affected my decision but the code needed more functions and it got more complicated
* How did the extension affect your design?
  + It got more complicated and longer
* Describe your tests (e.g., what you tested, equivalence classes).
  + With the test I tested the main functions of the game, at black-box I tested also the GUI point of view but in code I tested functions like the board or winning conditions
* What lessons did you learn from the comprehensive exercise (i.e., programming concepts, software process)?
  + We learned a process to design a game
  + How to test the code
* What functionalities are you going to demo?
  + We are going to test the game with two player mode and with the computer
* Who is going to speak about each portion of your presentation? (Recall: Each group will have ten minutes to present their work; minimum length of group presentation is seven minutes. Each student must present for at least two minutes of the presentation.)
  + We are going to brake it in half.
* Other notes:
  + <<Insert notes>>

<<Use your notes from above to complete create your slides and plan your presentation and demo.>>