March 20, 2019

Liam Aiello
Thamodh Egodawatte
Osama Hafez
Venura Perera
Kevin Subhash

C4: Documentation

Directory Structure

There is one main directory for our application. The *src/* directory contains all the files needed for the application. The files inside the *src/* directory include *c4_commands.py*, *c4.py*, *c4model.py*, *c4view.py*, *disk_commands.py*, *disk.py*, *player.py*, *point.py*. Our code used the Model-View-Controller (MVC) design pattern to make it easy to interact between classes and simplify the implementation. The Command design pattern was also implemented to keep track of all the moves the user makes onto the game board.

Code Structure

The *disk.py*, *player.py* and *point.py* are the base classes. These classes contain getter and setter methods for preliminary information. This preliminary information is used in the model for functionality purposes such as checking if the game is complete.

- The player.py class contains a *get_player_number(self*) method. This method is used in the model to know which player is dropping a disk onto the game board.
- The disk.py class contains a *get_point(self)* method. This method is used to get the point at which the disk is located on the board.
- The point.py class contains getter and setter methods for the x and y coordinates. This allows for easy implementation of methods that involve the disk as each disk has a unique (x, y) coordinate.

The *c4.py* class works as a hybrid for the View and the Controller in the MVC pattern. This class contains methods to make the buttons that are displayed on the view for the user to interact with. This class also contains the functions to run the start screen and help menu for the application. Lastly, this class contains the method that displays the game board on which the users play on.

def make_button(text, x, y, width, height, original_color, active_color, func):

This method creates a button for the user to click on. Upon mouse click each button has a different functionality. Additionally, when the user hovers over the button with their mouse, it will alter the colour slightly to indicate that the button can be pressed.

```
46  def make_button(text, x, y, width, height, original_color, active_colour, func):
47      user_mouse = pygame.mouse.get_pos()
48      user_click = pygame.mouse.get_pressed()
49
50      # if user hovers over button
51      if x + width > user_mouse[0] > x and y + height > user_mouse[1] > y:
52          pygame.draw.rect(screen_display, active_colour,(x, y, width, height))
53
54          if user_click[0] == 1 and func != None:
55              func()
56      else:
57          pygame.draw.rect(screen_display, original_color, (x, y, width, height))
58
59
60      small_text = pygame.font.Font("freesansbold.ttf", 20)
61      text_surface, text_rectangle = print_text(text, small_text)
62      text_rectangle.center = ((x + (width / 2)), (y + (height / 2)))
63      screen_display.blit(text_surface, text_rectangle)
```

```python
def start_screen():

    start = True

    while start:
        for event in pygame.event.get():
            # print(event)
            if event.type == pygame.QUIT:
                pygame.quit()
                quit()

        screen_display.fill(BLACK)

        large_text = pygame.font.Font('freesansbold.ttf', 64)

        text_surface, text_rectangle = print_text("C4: A Connect 4 Game", large_text)
        text_rectangle.center = ((SCREEN_WIDTH / 2),(SCREEN_HEIGHT / 2))
        screen_display.blit(text_surface, text_rectangle)

        # start button
        make_button("Start!", 150, 450, 100, 70, GREEN, LIGHT_GREEN, draw_stage)

        # help button
        make_button("Help", 350, 450, 100, 70, BLUE, LIGHT_BLUE, help_menu)

        # exit button
        make_button("Exit", 550, 450, 100, 70, RED, LIGHT_RED, game_quit)


        pygame.display.update()
        clock.tick(15)
```

def start_screen():

This method displays a start screen for the user to interact with. This function makes a call to the make_button(text, x, y, width, height, original_color, activie_color, func): function 3 times to make 3 buttons. These buttons are the Start, Help and Exit buttons that take the user to a different GUI upon mouse click.

The *draw_stage()* method is very similar to the s*tart_screen()* method as both methods display GUIs for the user to interact. The *draw_stage()* method contains the game board where users can drop displays and play the game.

The *c4model.py* class is where the logic behind the game is implemented. The main functions in this class include *game_over(self, disk)* and *insert_disk(self, column).*

- The *game_over(self, disk)* returns true if a user has won the game or if all the spots on the game board are filled with disks. The function works on the idea that if the disk inserted has three consecutive disk connected horizontally, vertically or diagonally. If there is 3 consecutive disks, then the disk inserted is the 4th one and therefore the game is over.
- The *insert_disk(self, column)* is a method to insert the user's disk onto the game board. The game board is defined by a list of lists where one list has 6 elements which are also lists of 7 elements with a 0 for an empty space, 1 for player 1's disk and 2 for player 2's disk.

```python
def insert_disk(self, column):
    """
    This function inserts a disk in the column that the
    player has chosen. It checks if there is space in
    the specified column and then creates a disk object in
    that position. If there is no space in the column, it
    will return false. If it inserts a disk object successfully,
    it returns true.
    """
    if (is_filled() == False): # Checks if board is not full
        for row in range (6,-1,-1): # Loops through rows to make sure there is an empty space to create a disk object
            if self.frame[row][column] == None:
                self.frame[row][column] = Disk(point(row, column), self.current_player)
                self.available_slots -= 1
                turn() # Changes turn to next player once disk is 'inserted'
                return True
    return False
```

def insert_disk(self, column):

Updates the game board with the new disk that the user placed.

def game_over(self, disk):

Checks for 3 consecutive disks upon disk insertion. If there are 3 consecutive disks either vertically, horizontally or diagonally, then the game is over and this function returns true.

```python
def game_over(self, disk):
    """
    This function checks if the disk that is inserted caused
    a connection of four disks. If so, this results in a win
    for the player. It checks if there is a row of four
    horizontally, vertically, or diagonally. It returns true
    if there is a connection of four indicating that the game
    is over. It returns false if there is no connection of four
    disks.

    """
    x_coord = disk.getPoint().getX() # Getting x and y coordinates of the passed in disk object
    y_coord = disk.getPoint().getY()
    consecutive_disks = [0,0,0,0,0,0,0,0]

    for shift in range (1, 4, 1): # Checking if there is a connection of four disks horizontally, vertically, or diagonally
        if(self.frame[x_coord+shift][y_coord] == disk):
            consecutive_disks[0] += 1
        if(self.frame[x_coord+shift][y_coord] == disk):
            consecutive_disks[1] += 1
        if(self.frame[x_coord][y_coord-shift] == disk):
            consecutive_disks[2] += 1
        if(self.frame[x_coord][y_coord+shift] == disk):
            consecutive_disks[3] += 1
        if(self.frame[x_coord+shift][y_coord+shift] == disk):
            consecutive_disks[4] += 1
        if(self.frame[x_coord+shift][y_coord-shift] == disk):
            consecutive_disks[5] += 1
        if(self.frame[x_coord-shift][y_coord+shift] == disk):
            consecutive_disks[6] += 1
        if(self.frame[x_coord-shift][y_coord-shift] == disk):
            consecutive_disks[7] += 1

    if(3 in consecutive_disks): # Returns true if there four disks are connected
        return True
    return False
```

As aforementioned, the Command design pattern was implemented in the application as well. The two major classes used to implement the Command design pattern are *c4_commands.py* and *disk_command.py.* This specific design pattern makes it very simple track the changes to the board as each command gets saved to a command queue.

In the *c4_commands.py* class, the major functions include: *add_command(self, command), remove_command(self, command), empty(self), operate_all(self)*

- The *add_command(self, command)* and *remove_command(self, command)* functions are self-explanatory as they add or remove a command from the command queue.
- The *empty(self)* function empties all the commands stored in the command queue (First In, First Out).
- The *operate_all(self)* function, executes all the operations in the command queue using a for loop.

The major function on the *disk_command.py* class is the *execute(self, column)* function. The major purpose of this function is to execute each command by returning true if the command is valid and false if the command is invalid.

Essentially, there is a for loop that loops through the 6 rows on the game board and checks to see if we can add the disk at that location. If the disk can be added at that location, then we add it in and decrement the available slots by 1. Finally we call the *turn()* method from the model to notify that a player has played their turn. This function returns false if the disk cannot be added at that specific location.

def execute(self, column):

Checks if we can execute a given command by adding the disk to the board and returning true. Otherwise, this function returns false.

```
12      def execute(self, column):
13          """
14          Create a disk with the given x,y coordinates and color.
15          Typically will call a strategy design pattern to actually create the
16          disk.
17          """
18
19          if (is_filled() == False):
20              for row in range (5,-1,-1):
21                  if self.frame[row][column] == None:
22                      self.frame[row][column] = Disk(point(row, column), self.current_player)
23                      self.available_slots -= 1
24                      self.model.turn() #how to call this as a function of model (so that player is
                            static)
25                      return True
26          return False
```

Extensions to the Game

To make the game user-friendly, we recommend playing multiplayer. Each user gets one turn, so after placing a disk onto the board, give control to your partner so they can place their disk. Once the board is filled or someone connected 4 disks in a row, the game is over.