a

Final Project Report for 2048

Alexander Wise

April 28, 2023

EECE 2140 – 04

Introduction:

Originally designed in 2014, 2048 is a puzzle game that utilizes squares initialized to the value or 2 or 4 with the end goal of creating a tile with the value of 2048. Tiles of the same value are combined into a new tile equal to their combined value with a new tile being generated after each keystroke. Tiles are moved up, down, left, or right to the border of the playing area. If the board completely fills up, the game is over. If the player achieves the tile, 2048, they win.

Application Design:

The main class in this version of 2048 is the Square class. Each tile is its own object and is initialized when generated.

```
def __init__(self, x_pos, y_pos, value = 0):
        self.x = x_pos
        self.y = y_pos
        if value == 0:
            self.value = self.value()
        else:
            self.value = value
```

Each object is initialized using it's randomly generated x_pos and y_pos in a separate array. Similarly, the value of the block is set as 0 by default and then initialized in a separate method.

```
def value(self):
        rand = random.random()
        if(rand <= 0.9):
            return 2
        else:
            return 4
```

As seen above, the value of the tile is either 2 or 4 with 2 having a 90% chance of being assigned and 4 having a 10% chance of being assigned. When two squares are equal to the same value, their value is assigned in the constructor.

```
def change_position(self, x, y):
```

```
    self.x = x
    self.y = y

def get_X_pos(self):
    return self.x

def get_Y_pos(self):
    return self.y
```

Above are the three location methods. Change_position is called whenever an arrow key is used and will change the x and y position of the object to be read by the function for the board so that change in location can be reflected on the GUI. get_X_pos and get_Y_pos are used to show where the value is in the GUI.

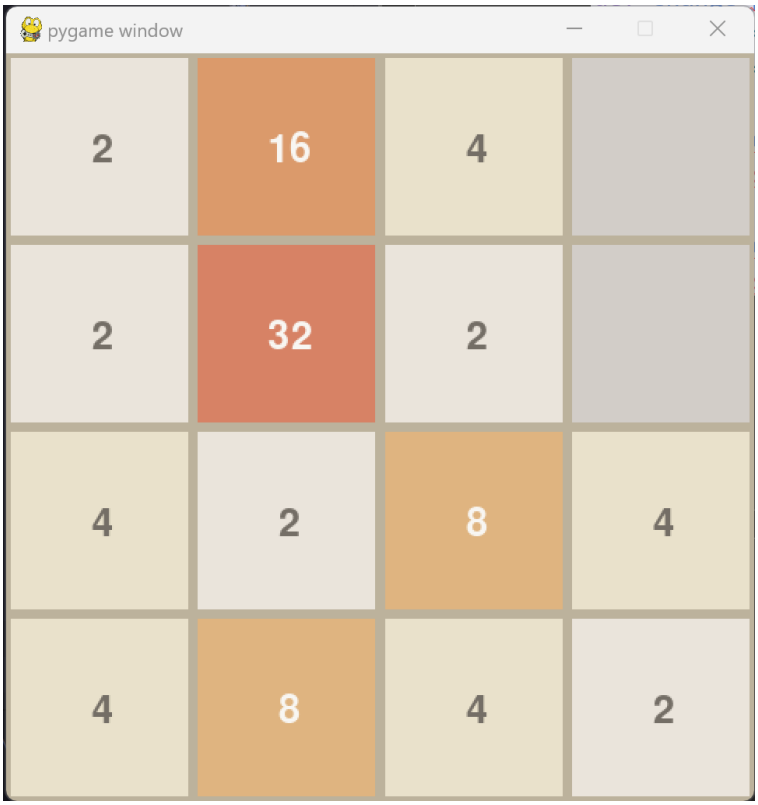

Figure 1: 2048 Board with tiles

The board itself is a screen of 480 x 480 pixels and each box is positioned at its array location * 120 + 3 in both the x and y direction.

When two Squares are of the same value, they are added together using __add__:

```
def __add__(self, other):
        return Square(self.x, self.y, self.value + other.value)
```

Creating a new object at the location of the tile closest to the wall depending on the arrow that is pressed and assigning a new value of the sum of the two squares that are combined. This check is completed when using the get_value(self) function in the Square class that will return the value of the object.

This check to see if the values are the same is done at the same time as the tile movement through the following code:

```
for col in range(len(board)):
        limit = 0
        previousVal = 0
        for row in range(len(board[col])):
           if(isinstance(board[row][col], Square)):
               temp = board[row][col]

               board[row][col] = None
               board[limit][col] = temp
               board[limit][col].change_position(col, limit)

               if previousVal == board[limit][col].get_value():
                   temp2 = board[limit - 1][col] + board[limit][col]
                   board[limit - 1][col] = temp2
                   board[limit][col] = None
                   previousVal = 0
                   limit -= 1
               else:
                   previousVal = board[limit][col].get_value()

               limit += 1
```

After the tile is moved, the code checks to see if that moved piece is equal to the piece directly next to it and if so, it will combine the two tiles together and create a new object of the combined values.

Using the pygame module, I was able to create a window, and add boxes to the window which the tiles would sit in. The window itself was initialized using the following code:

```
    pg.init()
    global SCREEN
    SCREEN = pg.display.set_mode((WINDOW_HEIGHT, WINDOW_WIDTH))
```

```
generate_box()
generate_box()
while True:
    SCREEN.fill((210,205,200,255))

    draw_grid()
    draw_box()

    for event in pg.event.get():
        if event.type == pg.QUIT:
            pg.quit()
        get_key_press()
    pg.display.update()
```

This code created a 480x480 window and populated it with the divider lines and boxes that the player interacts with. pg.quit() will end the program when the player closes the window.

Drawing the boxes themselves was done with the following code:

```
font = pg.font.Font('freesansbold.ttf', 25)
for row in range(len(board[0])):
    for col in range(len(board)):
        if isinstance(board[row][col], Square):
            box_color, text_color = setColor(board[row][col].get_value())

            rect = pg.Rect(board[row][col].get_X_pos() * 120 + 3,
            board[row][col].get_Y_pos() * 120 + 3, 114, 114)
            pg.draw.rect(SCREEN, box_color, rect)

            text = font.render(f"{board[row][col]}", True, text_color, None)
            textRect = text.get_rect()
            textRect.center = (board[row][col].get_X_pos() * 120 + 3 + 58.5,
               board[row][col].get_Y_pos() * 120 + 3 + 58.5)

            SCREEN.blit(text, textRect)
```

This code goes through each object in the array and if the selected position is an object of square, it will draw a box at its location and display its point value in the center of the box.

Instructions:

To play 2048, the user must run the code and use the arrow keys. Using the down key will shift all blocks down and combine any blocks that are the same value. This is the same for all four arrow key directions. The end goal is to create a box with the value of 2048. If a player reaches 2048, the program will display a message stating that you won and the player will have to close the program to end the game. Similarly, if there are no more open squares and no possible moves, the game will also end.

Libraries and Tools:

The libraries used in this program were pygame (graphics and windows, responsible for the GUI), random (for generating new tiles and placing them in random locations), and time (used for pausing the program after a game over or win phase.

Lessons Learned:

I learned more about object-oriented programming as I had to create and manipulate multiple object at the same time. I also learned about how to use pygame for future game or graphics design in python. I also had to do some deep planning for this program to calculate when to add two tiles together and how to correctly move them to their respective locations.

Screenshots of program:

was too dificult to reach the “You Win” screen so no screenshot will be shown.