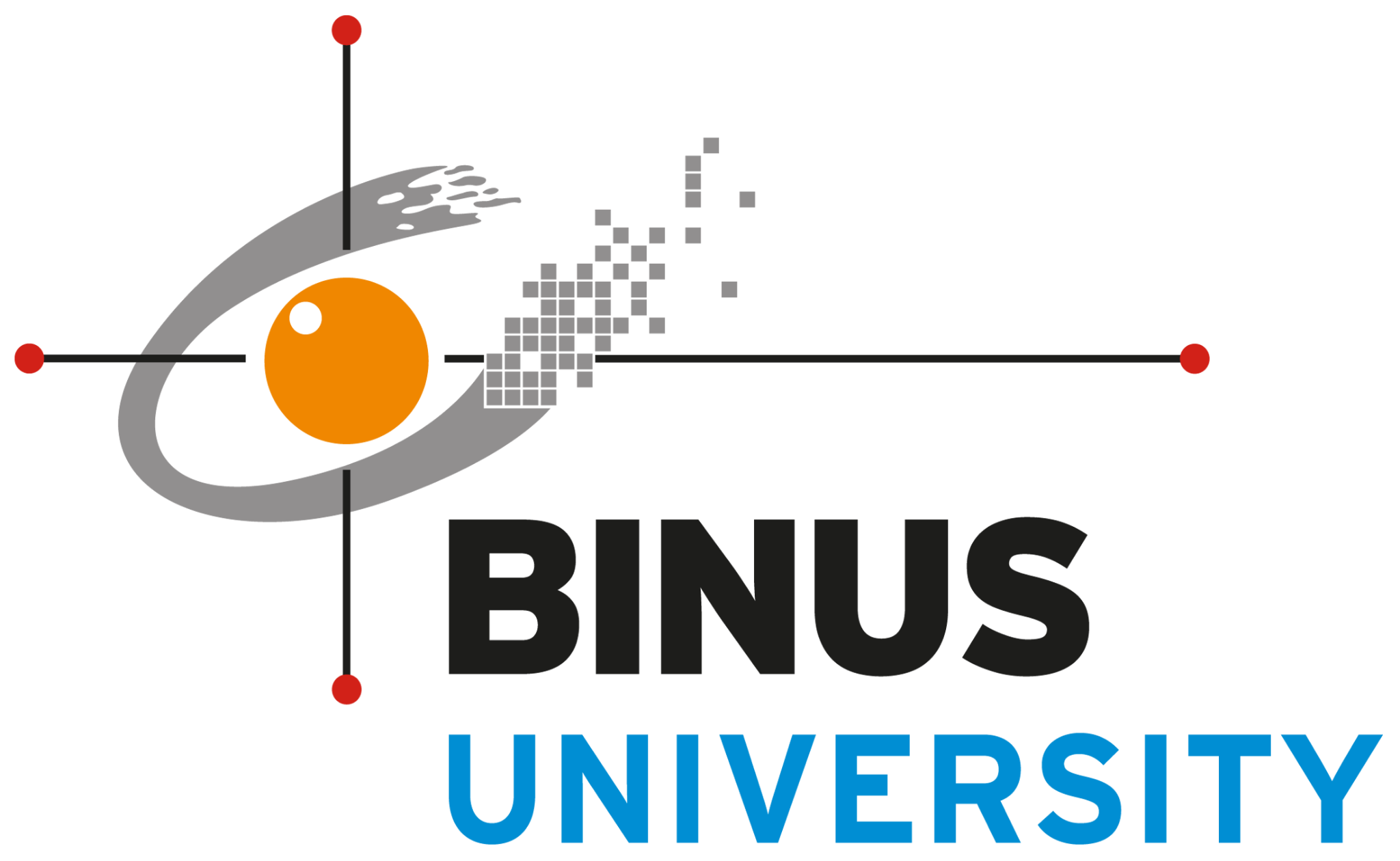
AlgoPro Report

Final project



**Andrew Sebastian Sibuea**

NIM 2602169711

L1BC

# **INTRODUCTION**

2048 is a single-player puzzle game where the goal is to combine numbered tiles on a grid to create a tile with the number 2048. The game starts with a 4x4 grid with two randomly placed tiles, each with the number 2. On each turn, the player can move all tiles in one of the four directions (up, down, left, right) and when two tiles with the same number touch each other, they merge into one tile with the value of the sum of the two tiles. The game is won when a tile with the number 2048 is created, and the game is lost when there are no more possible moves. The game is known for its simple yet addictive gameplay, as well as it's potential to teach problem-solving and logical thinking.

I was drawn to the game’s simple yet challenging gameplay, as well as it’s potential to teach important skills such as problem-solving and logical thinking. The concept of my project was inspired by the popular puzzle game '2048', and I utilized to code and resources from online sources as a starting point. However, I made significant modifications and added my own code to the project to make it unique. This included changes to the game mechanics, user interface, and overall design.

# **LINKS**

Link to my repository: <https://github.com/Andrew191103/AndrewAlgoFinal>

Link to the demo video: <https://drive.google.com/file/d/1qrMPX1bXynxj5XMjrKju_sBxdLyXhprS/view?usp=sharing>

# **MODULES**

* **import pygame:** imports the Pygame library, which is a library for making games and other multimedia applications with Python.
* **import random:** imports the random library, which is used for generating pseudo-random numbers.
* **from pygame import mixer:** imports the mixer module from the Pygame library, which is used for loading and playing sounds.
* **import button:** imports a library that you created to handle button functionality in the game, it is not part of the python standard library.
* **pygame.init():** initializes Pygame library
* **WIDTH = 400 and HEIGHT = 500:** sets the width and height of the game window
* **screen = pygame.display.set\_mode([WIDTH, HEIGHT]):** creates the game window with the specified width and height using the Pygame library
* **pygame.display.set\_caption('Andrews 2048'):** sets the title of the game window to "Andrews 2048"
* **timer = pygame.time.Clock():** creates a timer variable that will be used to control the frame rate of the game
* **fps = 60:** sets the frame rate of the game to 60 frames per second
* **font = pygame.font.Font('freesansbold.ttf', 24):** sets the font for the text in the game window to "freesansbold.ttf" with a size of 24
* **colors:** a dictionary containing RGB values for different colors used in the game, each key in the dictionary corresponds to a tile value (e.g. 2, 4, 8, etc.)
* **start\_img = pygame.image.load('start.png').convert\_alpha():** loads the start button image using the Pygame library's image module
* **exit\_img = pygame.image.load('exit.png').convert\_alpha():** loads the exit button image using the Pygame library's image module
* **start\_button = button.Button(85, 150, start\_img, 0.4):** creates an instance of the button class with the start button image and it's position on the screen
* **exit\_button = button.Button(85, 300, exit\_img, 0.4):** creates an instance of the button class with the exit button image and it's position on the screen
* **gamerun = False:** variable to check if the game is running
* **run = True:** variable to check if the main loop of the game is running
* **while run::** starts the main loop of the game, which continues to execute as long as the variable "run" is set to True
* **screen.fill((202, 228, 241)):** sets the background color of the game window
* **if start\_button.draw(screen)::** checks if the start button is clicked, if so the text "START" will be printed to the console
* **if exit\_button.draw(screen)::** checks if the exit button is clicked, if so the text "EXIT" will be printed to the console
* **for event in pygame.event.get()::** event handler for the game, it checks for all the events that are triggered
* **if event.type == pygame.QUIT::** if the QUIT event is triggered, the game will exit
* **if event.type == pygame.KEYUP::** if a key is released, it checks for the key pressed
* **if event.key == pygame.K\_UP::** if the up arrow key is pressed, the direction variable is set to 'UP'
* **elif event.key == pygame.K\_DOWN::** if the down arrow key is pressed, the direction variable is set to 'DOWN'
* **elif event.key == pygame.K\_LEFT::** if the left arrow key is pressed, the direction variable is set to 'LEFT'
* **elif event.key == pygame.K\_RIGHT::** if the right arrow key is pressed, the direction variable is set to 'RIGHT'
* **if game\_over::** if the game is over, it checks for the return key press
* **if event.key == pygame.K\_RETURN::** if the return key is pressed, the game variables are initialized and the game restarts
* **\_\_init\_\_(self, x, y, image, scale):** Initializes the button object with the x and y coordinates, the image, and the scale of the button. The function also sets the width and height of the button to the width and height of the image. It also sets the rect of the image, the top left coordinates of the image, and clicked status of the button to false.
* **draw(self, surface):** This function is used to draw the button on the surface. It gets the position of the mouse, checks if the mouse is over the button and if the button is clicked, and sets the action of the button to True if it's clicked. It also sets the clicked status of the button to false when the mouse button is released. It then draws the button on the screen. The function returns the action of the button.

# **ESSENTIAL ALGORITHMS**

* The main game loop continually updates the screen and handles user input, game logic, and drawing of the game elements.
* The "spawn new pieces" algorithm, generates new tiles on the board with a probability of a 2 or 4 value, and checks for game-over conditions.
* The "take turn" algorithm, which moves the tiles on the board in the direction specified by the user input, checks for and combines identical tiles, and updates the score.
* The "game over" algorithm, checks if the board is full and no more moves can be made and display the game over message and options to restart the game.
* The "high score" algorithm, keeps track of the highest score achieved during the game and writes it to a file for future reference.

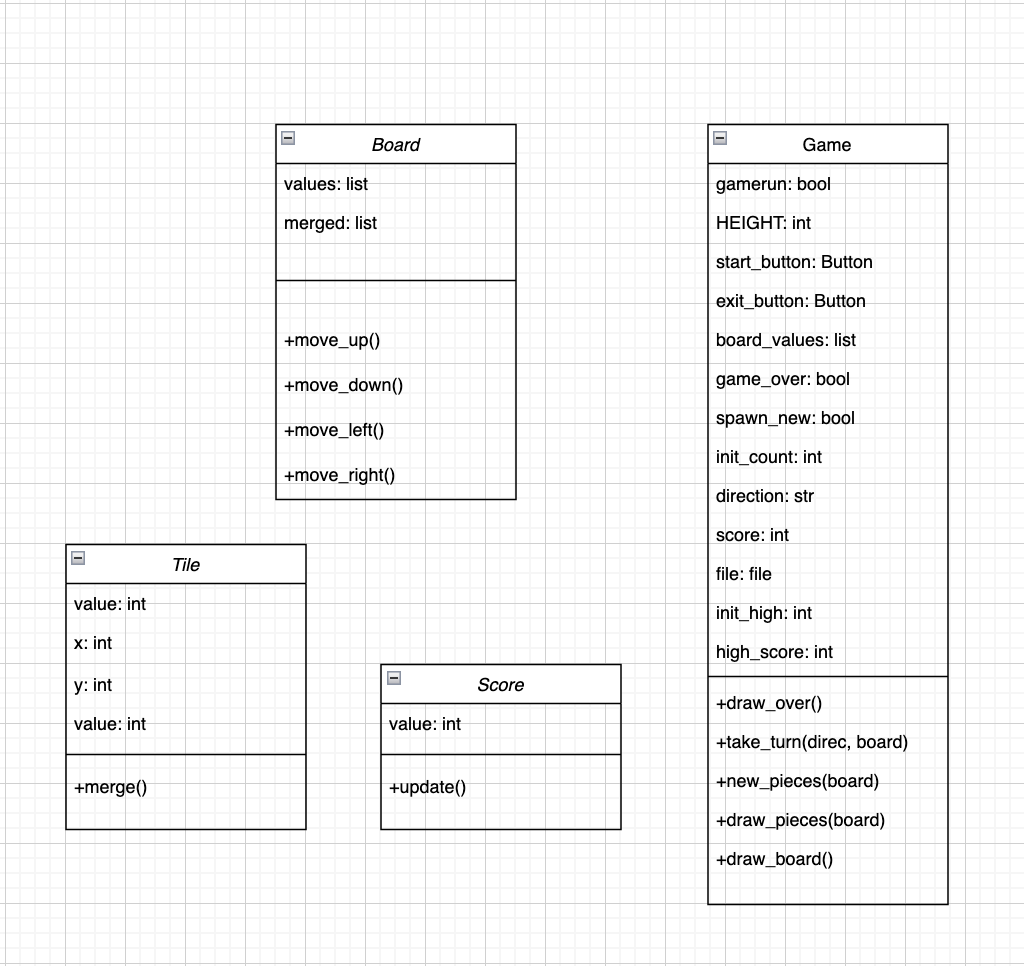
The code is a game loop for a game called "Andrews 2048" that is built using the Pygame library in Python. The loop runs while the variable "gamerun" is set to True. Within the loop, the game state is updated by calling several functions such as "draw\_board()" which is responsible for drawing the game board on the screen. Other functions such as "draw\_pieces()" and "new\_pieces()" are responsible for placing new pieces on the board and drawing them on the screen. The code also handles user input by monitoring events in the Pygame event queue, and updates the game state based on the user's actions. When the game is over, the code calls the "draw\_over()" function to display a message on the screen. The code also keeps track of the score and checks if it is higher than the high score, and updates the high score if necessary.

# **IMPLEMENTATION**

1. **Use-case Diagram**

# 

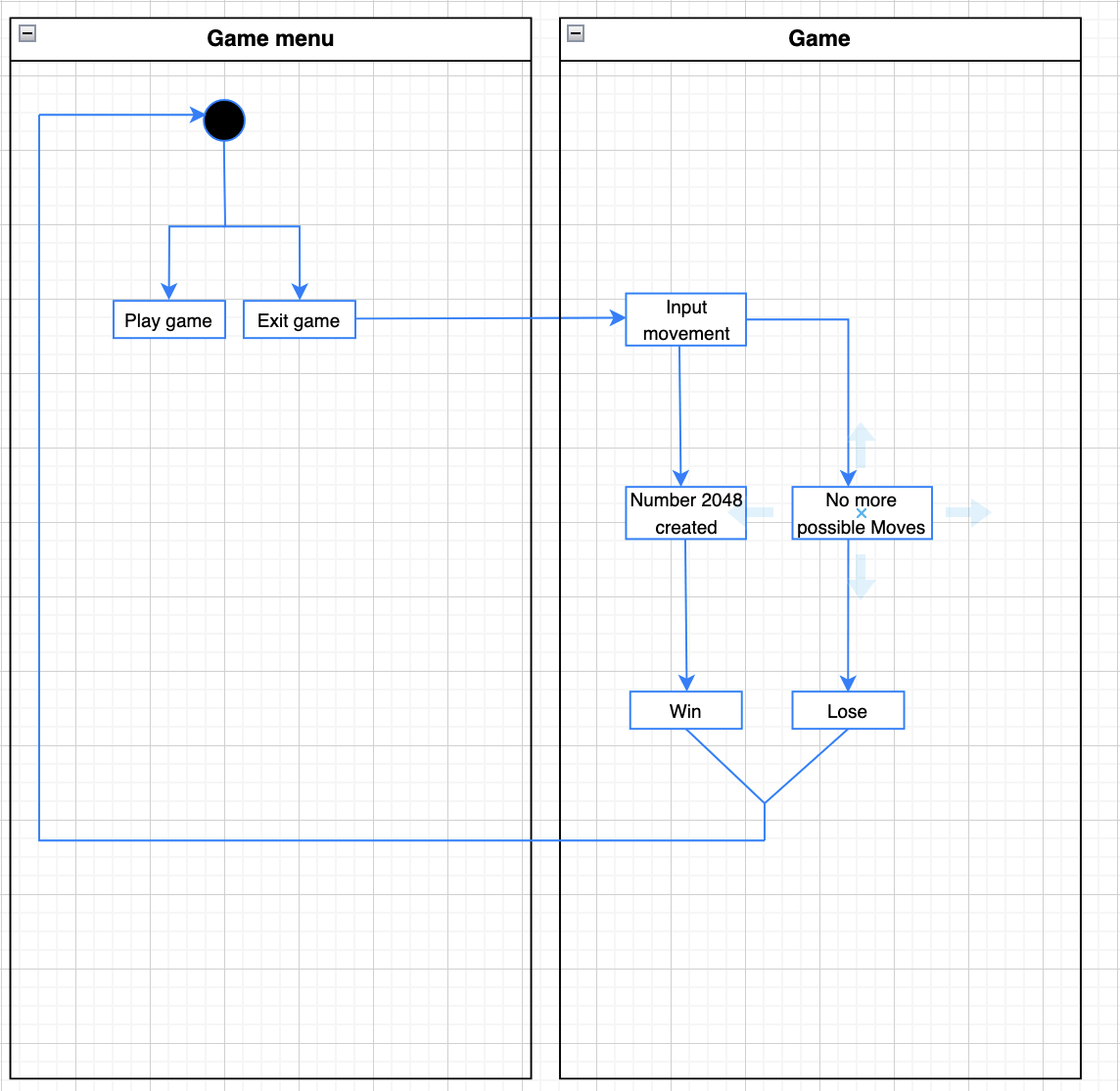
1. **Class Diagram**



**Relationships**

* The game class is the main class that controls the overall flow of the game and handles input from the player.
* The board class contains the logic for the game board and updates the state of the tiles on the board.
* The tile class contains the logic for each individual tile on the board, including its value and color.
* The score class keeps track of the player's score throughout the game and updates the high score.
* The game class contains a board object, which in turn contains multiple tile objects. The game class also contains a score object. The classes interact with each other to update the game state, handle user input, and display the game on the screen.

1. **Activity Diagram**

****

# **LESSON LEARNED/REFLECTION**

Creating the 2048 game using the Pygame library was a great learning experience. Pygame is a powerful library that allows developers to easily create 2D games using Python. It provides various modules for handling different aspects of game development such as graphics, sound, and input handling. One of the main advantages of using Pygame is that it is a built-in library, which eliminates the need for installing additional dependencies.

One of the biggest challenges I faced while creating this game was debugging and fixing errors. Pygame uses a lot of event-based programming, which can be tricky to understand and debug. Additionally, the game loop in Pygame is quite complex and requires a good understanding of how the library works.

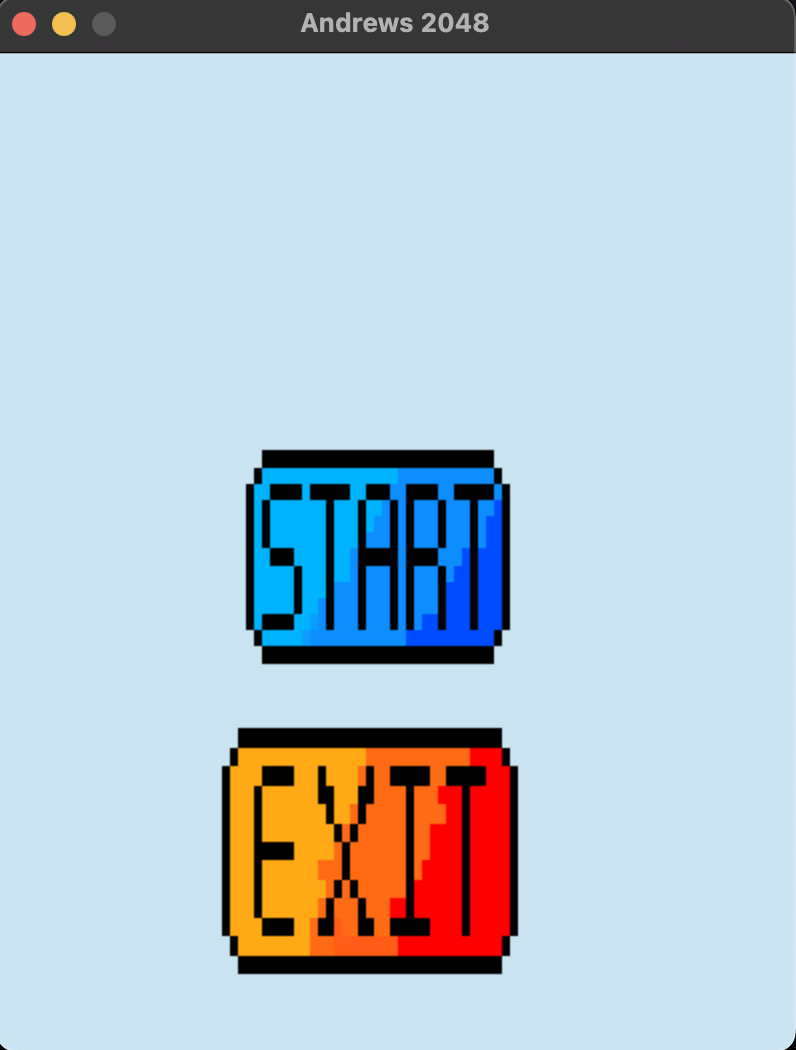
I also faced an issue with loading the button images and converting the alpha channel, it was a small issue but it took some time to debug and find the solution. Overall, creating the 2048 game using Pygame was a great learning experience. It helped me gain a deeper understanding of how to use the library and how to debug and fix errors that may arise during game development. I learned about the different modules and functions provided by Pygame and how to use them effectively to create a game. It is important to have a clear understanding of the game mechanics and the overall structure of the game before starting to write code. It is also crucial to organize the code in a way that makes it easy to understand and maintain. Additionally, I also learned about game development concepts such as game loops, event handling, and game states. In conclusion, creating the 2048 game using the Pygame library was a valuable learning experience that taught me about the power and capabilities of the library, as well as the importance of planning and organization in game development.

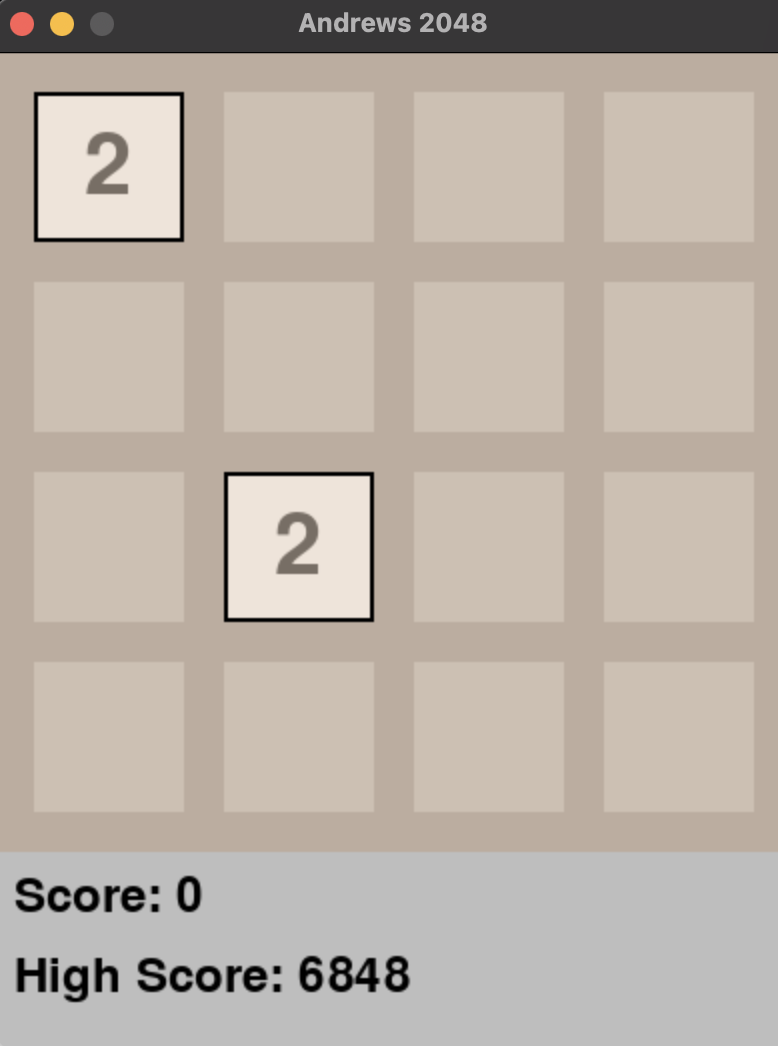
# 

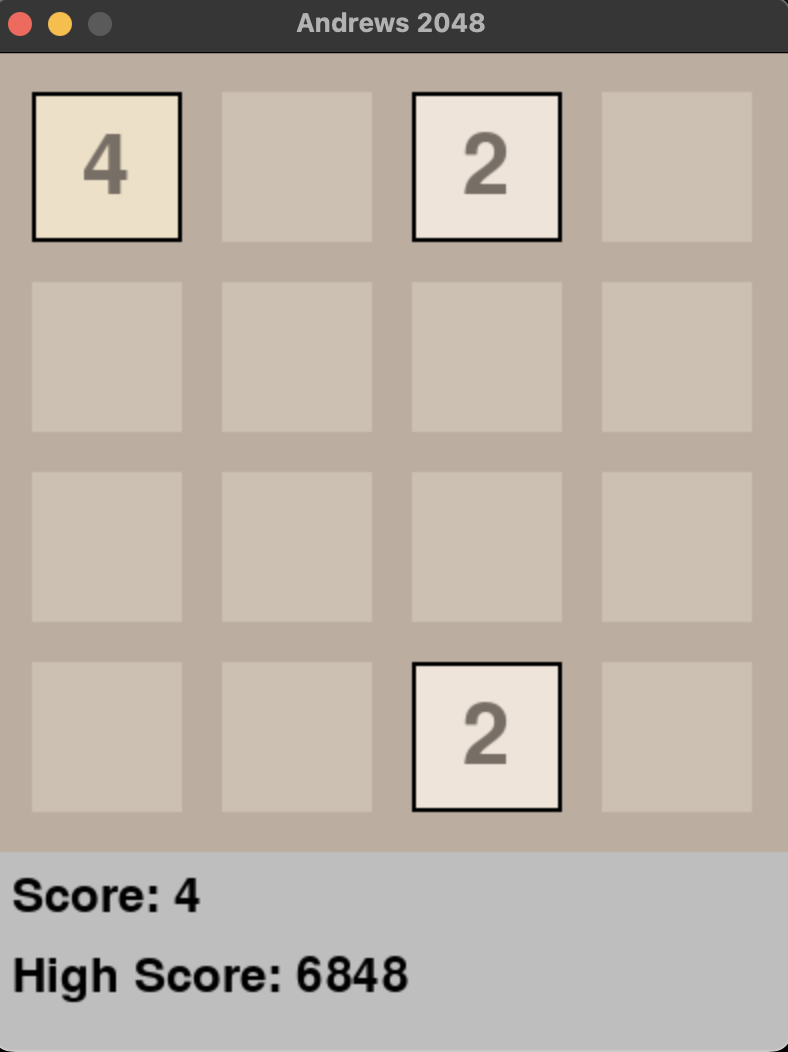
# 

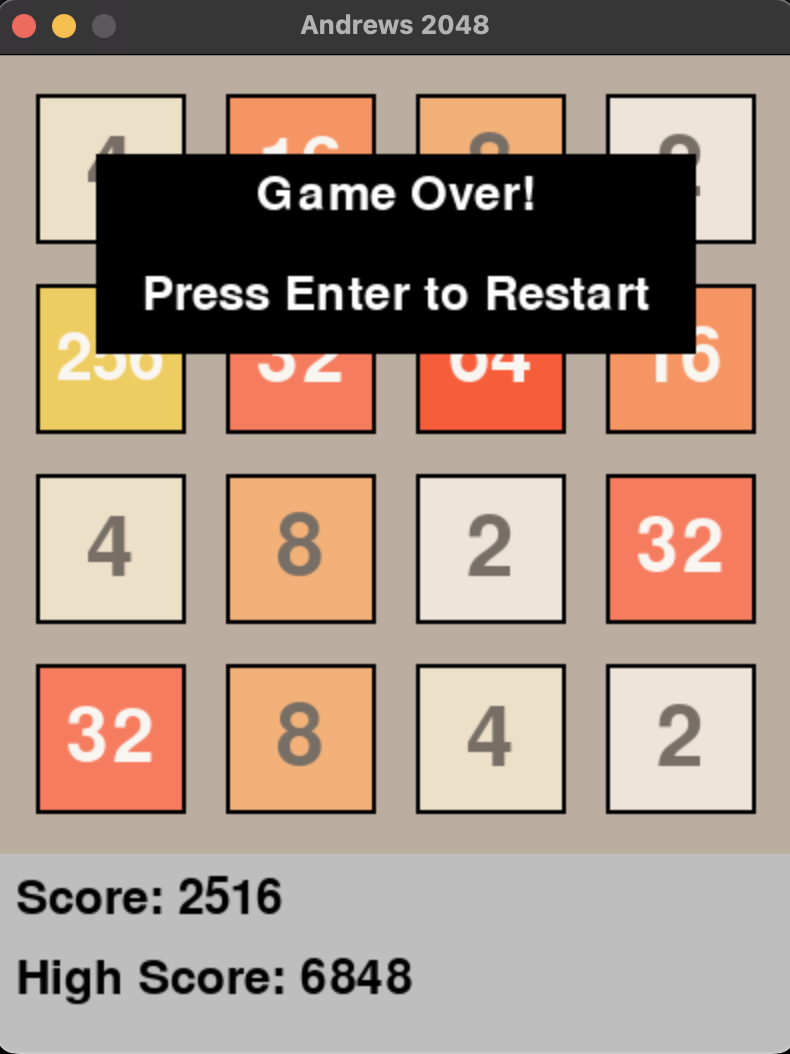
# 

# **SCREENSHOTS OF GAME**

This is the menu section/screen where the player can either start or quit the game.

When the player starts the game two boxes of “2”s will be randomly placed on the board also the background music will be played. 

Throughout the game, the player will have to somehow combine all of the boxes until it turns “2048”

Here the player loses because there is no more possible move, and can reset the game by clicking the enter button. 

# 

# 

# 