File - main.py

```python
import os
os.environ['PYGAME_HIDE_SUPPORT_PROMPT'] = "hide"

import gameclass


# run_game creates a game object and runs the game
def run_game():
    new_game = gameclass.Game()
    new_game.run_game()


if __name__ == '__main__':
    run_game()

```

File - helpers.py

```python
import pygame
import random

# Constants for game mechanics
MAX_DIFFERENCE = 255 // 2
MIN_DIFFERENCE = 4
MAX_BLOCKS = 5
MIN_BLOCKS = 2
COLOR_CHANGE_LEVELS = 2
BLOCK_CHANGE_LEVELS = 15


# random_color returns a randomly initialized pygame color within a given rgb range
def random_color(min_r=0, max_r=255):
    return pygame.Color(random.randint(min_r, max_r), random.randint(min_r, max_r), random.randint(min_r, max_r))


# get_high_score returns the highest score from a list of scores
def get_high_score(scores):
    if len(scores) == 0:
        return 0
    scores.sort()
    return scores[-1]

```

File - failmode.py

```
 1 import pygame
 2 import modeclass
 3 import blockarrayclass
 4 import helpers
 5
 6
 7 # FailMode is a class that contains the display for a game failure
 8 class FailMode(modeclass.Mode):
 9     def __init__(self, canvas):
10         super().__init__(canvas)
11
12     # run displays the fail screen with the most recent user score
13     def run(self, scores):
14         self.run_mode = True
15         menu_color = helpers.random_color(0, helpers.MAX_DIFFERENCE)
16         self.block_array = blockarrayclass.BlockArray(self.canvas, 2, menu_color,
   helpers.MAX_DIFFERENCE)
17
18         while self.run_mode:
19             self.block_array.display_blocks()
20             self.display_text("You have selected the wrong square", 32, -100)
21             self.display_text("Score: " + str(scores[-1]), 32, -50)
22             self.display_text("Click the different color to return to the menu", 24,
   100)
23             self.check_events(pygame.event.get())
24             pygame.display.flip()
25
26         return scores
27
28     # check_events determines if the correct block was clicked
29     # if so, we stop the fail mode display
30     def check_events(self, events):
31         for event in events:
32             if event.type == pygame.MOUSEBUTTONDOWN and self.block_array.
   check_mouse_press():
33                 self.clear_screen()
34                 self.run_mode = False
35         super().check_events(events)
36
```

File - menumode.py

```
import pygame
import copy
import modeclass
import blockarrayclass
import helpers

MAX_DIFFERENCE = 255//2


# MenuMode is a class that contains the menu mode
class MenuMode(modeclass.Mode):
    def __init__(self, canvas):
        super().__init__(canvas)

    # run displays the menu screen
    def run(self, scores):
        self.run_mode = True
        menu_color = helpers.random_color(0, MAX_DIFFERENCE)
        self.block_array = blockarrayclass.BlockArray(self.canvas, 2, menu_color, MAX_DIFFERENCE)

        while self.run_mode:
            self.block_array.display_blocks()
            self.display_text("Color Selector Game", 52, -100)
            self.display_text("Select the square that has a different color", 24, 100)
            self.display_text("Select the different color to start", 18, 125)
            self.display_text("High Score: " + str(helpers.get_high_score(copy.deepcopy(scores))), 32, -50)
            self.check_events(pygame.event.get())
            pygame.display.flip()

        return scores

    # check_events checks to make sure the mouse is pressed on the correct square
    # if this is the case, the menu will close
    def check_events(self, events):
        for event in events:
            if event.type == pygame.MOUSEBUTTONDOWN and self.block_array.check_mouse_press():
                self.clear_screen()
                self.run_mode = False
        super().check_events(events)

```

File - playmode.py

```
 1 import copy
 2 import pygame
 3 import modeclass
 4 import blockarrayclass
 5 import helpers
 6
 7
 8 # PlayMode is a class that contains the play mode
 9 class PlayMode(modeclass.Mode):
10     def __init__(self, canvas):
11         super().__init__(canvas)
12         self.level = 0
13         self.scores = []
14
15     # run creates a new block array on screen and
16     # keeps track of the current player score
17     def run(self, scores):
18         self.run_mode = True
19         self.scores = copy.deepcopy(scores)
20         self.set_calculated_block_array()
21
22         while self.run_mode:
23             self.block_array.display_blocks()
24             self.check_events(pygame.event.get())
25             pygame.display.flip()
26         return self.scores
27
28     # check_events increases the score if the user selects the correct square
29     # otherwise, the play mode is ended
30     def check_events(self, events):
31         for event in events:
32             if event.type == pygame.QUIT:
33                 pygame.quit()
34                 exit()
35             if event.type == pygame.MOUSEBUTTONDOWN:
36                 if self.block_array.check_mouse_press():
37                     self.set_calculated_block_array()
38                     self.level += 1
39                 else:
40                     self.scores.append(self.level)
41                     self.level = 0
42                     self.run_mode = False
43                 self.clear_screen()
44         super().check_events(events)
45
46     # set_calculated_block_array returns a block array with an increasingly difficult
    color and number of blocks
47     # The color difference diminishes with the level and the block size increases at
   helpers.BLOCK_CHANGE_LEVELS
48     def set_calculated_block_array(self):
49         play_color = helpers.random_color()
50         calculated_diff = max(helpers.MAX_DIFFERENCE - self.level * helpers.
   COLOR_CHANGE_LEVELS, helpers.MIN_DIFFERENCE)
51         calculated_num_blocks = min(helpers.MIN_BLOCKS + self.level // helpers.
   BLOCK_CHANGE_LEVELS, helpers.MAX_BLOCKS)
52         self.block_array = blockarrayclass.BlockArray(self.canvas,
   calculated_num_blocks, play_color, calculated_diff)
53
```

File - gameclass.py

```python
import pygame
import menumode
import playmode
import failmode


# Constants that hold the game state
MENU = 0
PLAY = 1
FAIL = 2
NUM_STATES = 3


# Game defines a game object that manages the game state
# and controls game mechanics
class Game:
    def __init__(self):
        # initialize game variables
        self.exit = False
        self.canvas = pygame.display.set_mode((500, 500))
        self.game_state = MENU
        self.scores = []
        self.states = {
            MENU: menumode.MenuMode(self.canvas),
            PLAY: playmode.PlayMode(self.canvas),
            FAIL: failmode.FailMode(self.canvas)
        }

        # initialize game settings
        pygame.init()
        pygame.display.set_caption("Color Selector")

    # run_game sets the game loop and starts the display
    def run_game(self):
        while not self.exit:
            self.run_display()

    # run_display selects the display based on the game state
    # The states are in circular order:
    #   MENU
    #   PLAY
    #   FAIL
    def run_display(self):
        self.scores = self.states[self.game_state].run(self.scores)
        self.game_state = (self.game_state + 1) % NUM_STATES


```

File - modeclass.py

```
 1 import pygame
 2 import blockarrayclass
 3
 4
 5 # Mode class is the generic class for each game mode
 6 # it defines the shared functions between each mode
 7 class Mode:
 8     def __init__(self, canvas):
 9         super().__init__()
10         self.canvas = canvas
11         self.block_array = blockarrayclass.BlockArray(self.canvas)
12         self.run_mode = True
13
14     # run displays the game mode
15     def run(self, scores):
16         return scores
17
18     # check events checks to see if the events have been processed
19     def check_events(self, events):
20         for event in events:
21             if event.type == pygame.QUIT:
22                 pygame.display.quit()
23                 pygame.quit()
24                 exit(0)
25
26     # display_text displays white text that is centered on screen
27     # the height and size of the text can be modified
28     def display_text(self, t="", text_size=32, height=0):
29         font = pygame.font.SysFont(None, text_size)
30         text = font.render(t, True, pygame.Color(255, 255, 255))
31         self.canvas.blit(text, (self.canvas.get_width() / 2 - text.get_width() / 2,
32                                 self.canvas.get_height() / 2 - text.get_height() / 2
    + height))
33
34     # clear screen fills the screen with black to remove all contents form the screen
35     def clear_screen(self):
36         self.canvas.fill(pygame.Color(0, 0, 0))
37
```

File - blockarrayclass.py

```python
import copy
import pygame
import random

BORDER = 3


# BlockArray creates an array of color blocks, selecting one block to be different
# This class draws the display for the array of blocks that gets created on screen
class BlockArray:
    def __init__(self, canvas, block_num=2, base_color=pygame.Color(0, 0, 0), color_diff=0):
        # initialize class variables for display
        self.canvas = canvas
        self.block_num = block_num
        self.base_color = base_color
        self.special_block = random.randint(0, self.block_num**2 - 1)
        self.color_diff = color_diff
        self.diffr, self.diffg, self.diffb = self.color_decomposition()

        # block characteristics
        canvas_width = self.canvas.get_width()
        canvas_height = self.canvas.get_height()
        self.block_width = canvas_width/self.block_num - BORDER
        self.block_height = canvas_height/self.block_num - BORDER

    # display_blocks draws the array of blocks on screen and selects the "special" block
    def display_blocks(self):
        for i in range(self.block_num):
            for j in range(self.block_num):
                x = i * (self.block_height + BORDER)
                y = j * (self.block_width + BORDER)

                display_color = self.base_color
                if i * self.block_num + j == self.special_block:
                    display_color = self.change_block_color()
                pygame.draw.rect(self.canvas, display_color, pygame.Rect(x, y, self.block_width, self.block_height))

    # change_block_color changes the color of the rgb block values based on the calculated difference
    def change_block_color(self):
        new_color = copy.deepcopy(self.base_color)
        new_color.r = self.color_val(new_color.r, self.diffr)
        new_color.g = self.color_val(new_color.g, self.diffg)
        new_color.b = self.color_val(new_color.b, self.diffb)
        return new_color

    # color val computes a difference without overflowing the rgb values
    def color_val(self, curr, diff):
        if curr + diff < 0 or curr + diff > 255:
            diff = -diff
        return curr + diff

    # color_decomposition calculates diffs for the reg green and blue values evenly distributed by the color_diff
    def color_decomposition(self):
        remaining_value = self.color_diff
        r = random.randint(0, remaining_value)
        remaining_value -= r
        g = random.randint(0, remaining_value)
        remaining_value -= g
        b = random.randint(0, remaining_value)
        remaining_value -= b
        return r, g, b

```

```
63      # check_mouse_press determines if the mouse press was on the correct square
64      # it will return true if so and false otherwise
65      def check_mouse_press(self):
66          x, y = pygame.mouse.get_pos()
67
68          i = self.special_block // self.block_num
69          j = self.special_block % self.block_num
70
71          min_x = i * (self.block_width + BORDER)
72          min_y = j * (self.block_height + BORDER)
73          max_x = min_x + self.block_width
74          max_y = min_y + self.block_height
75
76          if min_x < x < max_x and min_y < y < max_y:
77              return True
78          return False
79
```