

```
1 import os
2 os.environ['PYGAME_HIDE_SUPPORT_PROMPT'] = "hide"
3
4 import gameclass
5
6
7 # run_game creates a game object and runs the game
8 def run_game():
9     new_game = gameclass.Game()
10    new_game.run_game()
11
12
13 if __name__ == '__main__':
14     run_game()
15
```

```
1 import pygame
2 import random
3
4 # Constants for game mechanics
5 MAX_DIFFERENCE = 255 // 2
6 MIN_DIFFERENCE = 4
7 MAX_BLOCKS = 5
8 MIN_BLOCKS = 2
9 COLOR_CHANGE_LEVELS = 2
10 BLOCK_CHANGE_LEVELS = 15
11
12
13 # random_color returns a randomly initialized pygame color within a given rgb range
14 def random_color(min_r=0, max_r=255):
15     return pygame.Color(random.randint(min_r, max_r), random.randint(min_r, max_r),
16                             random.randint(min_r, max_r))
17
18 # get_high_score returns the highest score from a list of scores
19 def get_high_score(scores):
20     if len(scores) == 0:
21         return 0
22     scores.sort()
23     return scores[-1]
24
```

```

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,
17 helpers.MAX_DIFFERENCE)
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

```

```

1 import pygame
2 import copy
3 import modeclass
4 import blockarrayclass
5 import helpers
6
7 MAX_DIFFERENCE = 255//2
8
9
10 # MenuMode is a class that contains the menu mode
11 class MenuMode(modeclass.Mode):
12     def __init__(self, canvas):
13         super().__init__(canvas)
14
15     # run displays the menu screen
16     def run(self, scores):
17         self.run_mode = True
18         menu_color = helpers.random_color(0, MAX_DIFFERENCE)
19         self.block_array = blockarrayclass.BlockArray(self.canvas, 2, menu_color,
MAX_DIFFERENCE)
20
21         while self.run_mode:
22             self.block_array.display_blocks()
23             self.display_text("Color Selector Game", 52, -100)
24             self.display_text("Select the square that has a different color", 24, 100
)
25             self.display_text("Select the different color to start", 18, 125)
26             self.display_text("High Score: " + str(helpers.get_high_score(copy.
deepcopy(scores))), 32, -50)
27             self.check_events(pygame.event.get())
28             pygame.display.flip()
29
30         return scores
31
32     # check_events checks to make sure the mouse is pressed on the correct square
33     # if this is the case, the menu will close
34     def check_events(self, events):
35         for event in events:
36             if event.type == pygame.MOUSEBUTTONDOWN and self.block_array.
check_mouse_press():
37                 self.clear_screen()
38                 self.run_mode = False
39             super().check_events(events)
40

```

```

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
47     # color and number of blocks
48     # The color difference diminishes with the level and the block size increases at
49     # helpers.BLOCK_CHANGE_LEVELS
50     def set_calculated_block_array(self):
51         play_color = helpers.random_color()
52         calculated_diff = max(helpers.MAX_DIFFERENCE - self.level * helpers.
53             COLOR_CHANGE_LEVELS, helpers.MIN_DIFFERENCE)
54         calculated_num_blocks = min(helpers.MIN_BLOCKS + self.level // helpers.
55             BLOCK_CHANGE_LEVELS, helpers.MAX_BLOCKS)
56         self.block_array = blockarrayclass.BlockArray(self.canvas,
57             calculated_num_blocks, play_color, calculated_diff)
58
59

```

```
1 import pygame
2 import menu mode
3 import play mode
4 import fail mode
5
6
7 # Constants that hold the game state
8 MENU = 0
9 PLAY = 1
10 FAIL = 2
11 NUM_STATES = 3
12
13
14 # Game defines a game object that manages the game state
15 # and controls game mechanics
16 class Game:
17     def __init__(self):
18         # initialize game variables
19         self.exit = False
20         self.canvas = pygame.display.set_mode((500, 500))
21         self.game_state = MENU
22         self.scores = []
23         self.states = {
24             MENU: menu mode.MenuMode(self.canvas),
25             PLAY: play mode.PlayMode(self.canvas),
26             FAIL: fail mode.FailMode(self.canvas)
27         }
28
29         # initialize game settings
30         pygame.init()
31         pygame.display.set_caption("Color Selector")
32
33         # run_game sets the game loop and starts the display
34     def run_game(self):
35         while not self.exit:
36             self.run_display()
37
38         # run_display selects the display based on the game state
39         # The states are in circular order:
40         # MENU
41         # PLAY
42         # FAIL
43     def run_display(self):
44         self.scores = self.states[self.game_state].run(self.scores)
45         self.game_state = (self.game_state + 1) % NUM_STATES
46
47
```

```

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
33                                + height))
34
35    # clear screen fills the screen with black to remove all contents form the screen
36    def clear_screen(self):
37        self.canvas.fill(pygame.Color(0, 0, 0))

```

```

1 import copy
2 import pygame
3 import random
4
5 BORDER = 3
6
7
8 # BlockArray creates an array of color blocks, selecting one block to be different
9 # This class draws the display for the array of blocks that gets created on screen
10 class BlockArray:
11     def __init__(self, canvas, block_num=2, base_color=pygame.Color(0, 0, 0),
12         color_diff=0):
13         # initialize class variables for display
14         self.canvas = canvas
15         self.block_num = block_num
16         self.base_color = base_color
17         self.special_block = random.randint(0, self.block_num**2 - 1)
18         self.color_diff = color_diff
19         self.diff_r, self.diff_g, self.diff_b = self.color_decomposition()
20
21         # block characteristics
22         canvas_width = self.canvas.get_width()
23         canvas_height = self.canvas.get_height()
24         self.block_width = canvas_width/self.block_num - BORDER
25         self.block_height = canvas_height/self.block_num - BORDER
26
27         # display_blocks draws the array of blocks on screen and selects the "special"
28         # block
29         def display_blocks(self):
30             for i in range(self.block_num):
31                 for j in range(self.block_num):
32                     x = i * (self.block_height + BORDER)
33                     y = j * (self.block_width + BORDER)
34
35                     display_color = self.base_color
36                     if i * self.block_num + j == self.special_block:
37                         display_color = self.change_block_color()
38                     pygame.draw.rect(self.canvas, display_color, pygame.Rect(x, y, self.
39 block_width, self.block_height))
40
41         # change_block_color changes the color of the rgb block values based on the
42         # calculated difference
43         def change_block_color(self):
44             new_color = copy.deepcopy(self.base_color)
45             new_color.r = self.color_val(new_color.r, self.diff_r)
46             new_color.g = self.color_val(new_color.g, self.diff_g)
47             new_color.b = self.color_val(new_color.b, self.diff_b)
48             return new_color
49
50         # color_val computes a difference without overflowing the rgb values
51         def color_val(self, curr, diff):
52             if curr + diff < 0 or curr + diff > 255:
53                 diff = -diff
54             return curr + diff
55
56         # color_decomposition calculates diffs for the red green and blue values evenly
57         # distributed by the color_diff
58         def color_decomposition(self):
59             remaining_value = self.color_diff
60             r = random.randint(0, remaining_value)
61             remaining_value -= r
62             g = random.randint(0, remaining_value)
63             remaining_value -= g
64             b = random.randint(0, remaining_value)
65             remaining_value -= b
66             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
```