ADS BOARD GAME (HUES AND CUES)

PROJECT PRESENTATION

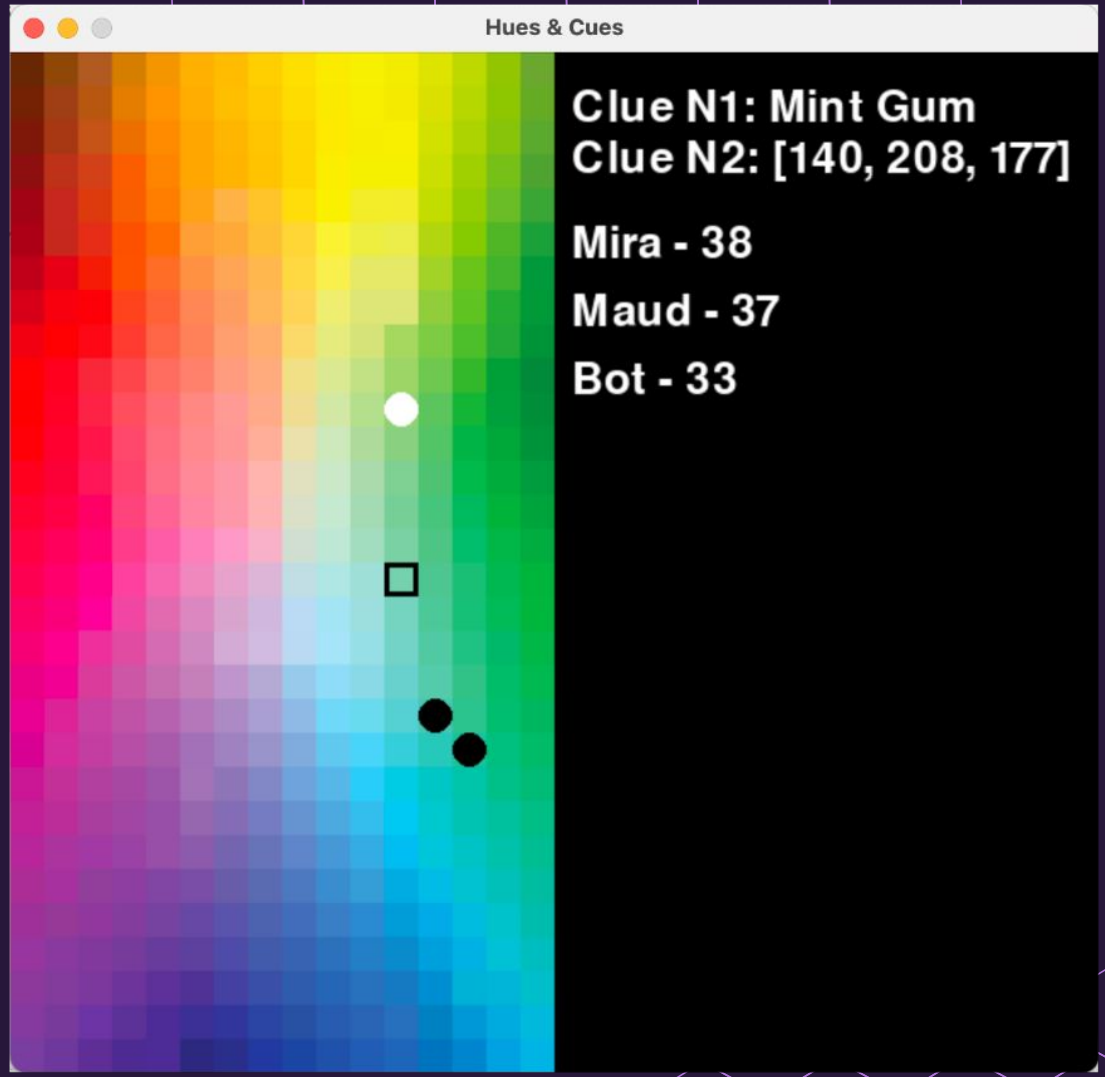
Mariana Berrio, Miranda Drummond, Sofia Moreno, Keti Sulamanidze

TABLE OF CONTENTS

01

INTRODUCTION

Overview of our project


Hues & Cues
Clue N1: Mint Gum
Clue N2: [140, 208, 177]
Mira - 38
Maud - 37
Bot - 33

02

WHY HUES AND CUES?

You can enter a subtitle here if you need it

Why?

1

Digitising a board game which is yet to be coded.

2

Simple logic, easy to make more efficient

3

Plays off of what we've learnt in class

4

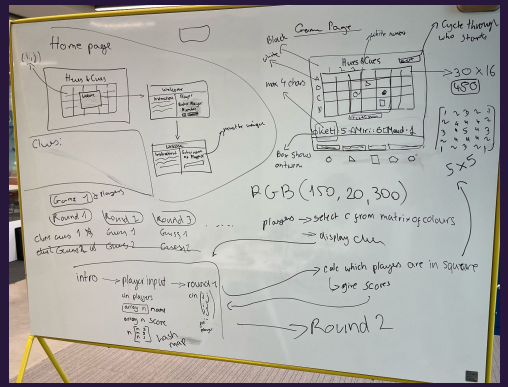
Fun game, interesting to implement

03

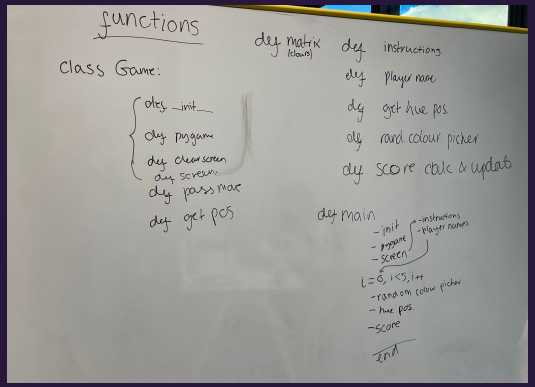
IMPLEMENTATION

WHITEBOARD -> CODE

PLANNING

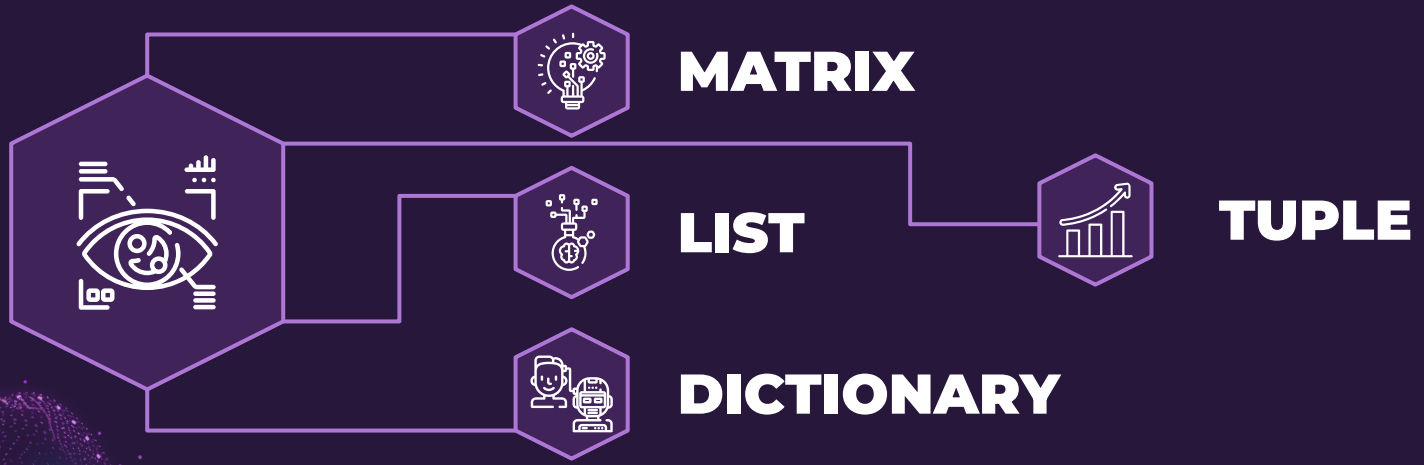


CODING




04
DATA STRUCTURES

DATA STRUCTURES

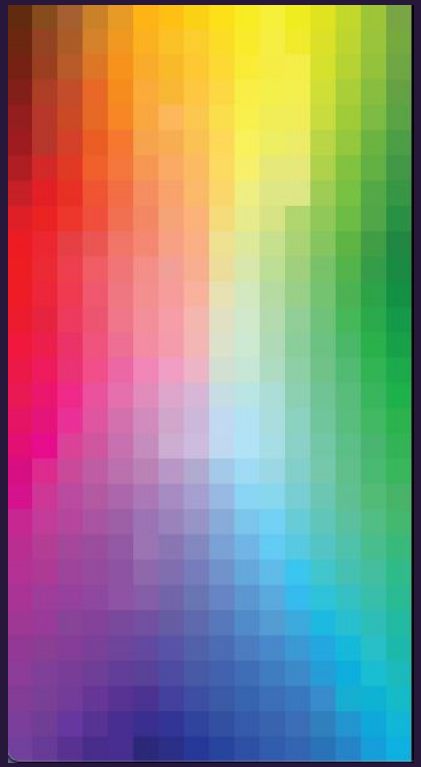


CLASSES UTILISED

- Class board:
  - Creates a cell matrix is a matrix of Cell objects

- Class cell:
  - Gets x and y coordinates of top left cell
  - Gets color in RGB format of each cell

MATRIX



- Matrix of cell objects
- Inside of rgb_matrix.py
- Time complexity: O(n*m)

LIST

- We implemented lists many times along the code. For example, we used them to be able to get an input of the number of players and their names and store them for each round

DICTIONARY (HASHMAP)

- Players and their scores

self.PLAYER_NAMES_SCORES = PLAYER_NAMES_SCORES
self.round_scores_hidden = round_scores_hidden

- Colour clues

E.g., : "Apple" : [[175, 30, 35], [198, 33, 39], [222, 31, 38], [237, 28, 35], [238, 29, 34], [239, 28, 37], [236, 27, 46], [240, 24, 60], [236, 24, 72], [213, 40, 40], [227, 32, 37],

[235, 35, 35], [236, 39, 47], [235, 42, 52], [237, 38, 59], [231, 36, 64], [233, 48, 37], [238, 54, 44],

[230, 64, 68], [237, 63, 78], [240, 79, 61], [245, 142, 97]]

TUPLE

Problem: RGB colours can´t be used as keys if they are inside a list

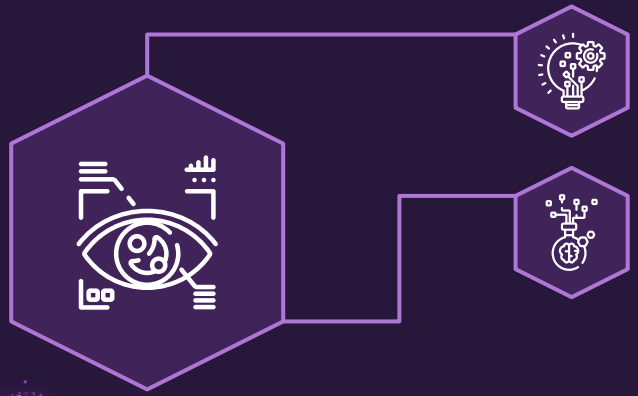
Solution:

```
for key in clue_to_rgb.keys():
        for index, rgb in enumerate(clue_to_rgb[key]):
                rgb = tuple(rgb)
                clue_to_rgb[key][index] = rgb
rgb_to_color_clue = {}

# 0(n*m)

for key in clue_to_rgb.keys():
        for rgb in clue_to_rgb[key]:
                rgb_to_color_clue[rgb] = key
```

05

ALGORITHMS

ALGORITHMS



Binary Search

QuickSort

QUICK SORT

```
def quicksort_inplace(player_seq_list, left, right):
        if left > right:
                return
        pivot = partition(player_seq_list, left, right)
        quicksort_inplace(player_seq_list, left, pivot - 1)
        quicksort_inplace(player_seq_list, pivot + 1, right)
```

O(plog(p))
p = number of players but p can be considered a constant so O(1)

BINARY SEARCH

When and how are we using it?

Board_rgb_matrix —> flattened_board_rgb_matrix

flattened_board_rgb_matrix = [[(r,g,b)(row1,col1)], [(r,g,b)(row2,col2)], [(r,g,b)(row3,col3)], ...]

# we sort it based on r,g,b (one after the other) and we keep it saved
flattened_board_rgb_matrix = sorted(flattened_board_rgb_matrix, key=lambda x: x[0])

And then we do a simple binary search

```
while right >= left:
    mid = (left+right)//2
    #compare if we found the whole tuple
    if flattened board rgb matrix[mid][0] == rgb_tuple:
        return flattened board rgb matrix[mid][1]
    elif flattened board_rgb_matrix[mid][0] > rgb_tuple:
        right = mid - 1
    else:
        left = mid + 1
#print("Could not find", rgb_tuple)
return (7, 15)
```

O(log(n))
n being the number of color cells in the color matrix

DRIVER CODE
Class Round

Main function endlessly creates a class Round and calls a function called play_round

```
class Round:
    def __init__(self, true_rgb, true_indexing, screen, player_seq_list, N_PLAYERS, board,
CELL_SIZE, PLAYER_NAMES_SCORES, round_scores_hidden, color_clue):
        self.true_rgb = true_rgb  # every round has a true rgb which is a random color chosen
by computer
        self.true_indexing = true_indexing  # the random colors indexing (position) in the
cell matrix
        self.screen = screen  # screen we draw on
        self.player_seq_list = player_seq_list  # ordered list of players
        self.N_PLAYERS = N_PLAYERS  # number of players
        self.board = board  # Board instance
        self.CELL_SIZE = CELL_SIZE  # size of cells in the board
        self.PLAYER_NAMES_SCORES = PLAYER_NAMES_SCORES  # dictionary of players and their
scores that appear on screen
        self.circle_locations = {}  # locations of players current guesses
        self.round_scores_hidden = round_scores_hidden  # dictionary of players scores which
are calculated instantly but not shown on display yet
        self.color_clue = color_clue  # clue for the color
```

DRIVER CODE
Class Round

What is the logic inside play_round function?

1. redraw the board and make sure everything is ready for a new round

```
# empty circles
self.empty circle locations()
#draw board before the round starts
self.draw_board_players_clue_circles_on_screen()
```

DRIVER CODE
Class Round

What is the logic inside play_round function?

2. select a player to make a move (from a quicksorted dictionary)

```
for i in range(self.N_PLAYERS):
    current_player = self.player_seq_list[i][0]
    self.print_whos_turn(current_player)
```

3. If player is a bot execute bots_move function

```
if current_player == 'Bot':
    time.sleep(1)
    # O(nlogn)
    self.bots_move()
    not_clicked_on_cell = False
```

```
# O(nlogn) n being total number of color cells
def bots_move(self):
    rand_rgb_index = random.randint( 0,
len(clue_to_rgb[self.color_clue])-1)
    bots_rgb_tuple = clue_to_rgb[ self.color_clue][rand_rgb_index]
    pos = binary_search(bots_rgb_tuple , 0,
len(flattened_board_rgb_matrix)- 1)

    self.circle_locations['Bot'] = (pos[0], pos[1])
    self.calculate_distance_from_real_color((pos[ 0], pos[1]),
'Bot')
```

DRIVER CODE
Class Round

4. If the player is not a bot we check if for a mouse click. while player hasn't clicked on valid cell (we call a function inside_board o(1) to check validity and check if any other player has already made a move there o(p)))

```
while not_clicked_on_cell:
    for event in pygame.event.get():
        if event.type == pygame.MOUSEBUTTONUP:
            pos = pygame.mouse.get_pos()
```

5. We also need to know that the click was on a valid cell (we call a function inside_board o(1) to check validity and check if any other player has already made a move there o(p)))

```
if self.inside board(pos) and (pos[0]//self.CELL_SIZE, pos[1]//self.CELL_SIZE) not in
self.circle_locations.values():
```

DRIVER CODE
Class Round

6. If valid -> put a circle -> calculate_distance_from_real_color -> put in hidden current scores in a hashmap

```
self.circle_locations[current_player] = (pos[0]//self.CELL_SIZE, pos[1]//self.CELL_SIZE)
self.calculate_distance_from_real_color((pos[0]//self.CELL_SIZE, pos[1]//self.CELL_SIZE),
current_player)
not_clicked_on_cell = False
```

7. Once the round is over actual cell and hidden scores are revealed to players and Round 2 begins.

```
self.update_scores()
self.display_actual_color()
time.sleep(4)
```

THANKS!

DO YOU HAVE ANY QUESTIONS?