SET Game Report

GitHub link: https://github.com/Ruohann/SET

Students number

Ruohan Yang: 7102887

Xujie Yang: 0754978

1| Orientation

1.1Background

SET is a real-time card game. It was created by Marsha Falco in 1974 and was not brought to market by Set Enterprises until 1991. The game's creation was somewhat serendipitous; while working in genetic research, Falco began using symbols to visualize complex genetic data on cards. Over time, these symbols inspired the mechanics of SET (Wikipedia contributors, 2024b).

1.2 Gameplay and Components

SET consists of 81 unique cards, each featuring one of three symbols (diamonds, squiggles, or ovals), which appear in varying numbers (one, two, or three), colors (red, purple, or green), and shadings (solid, striped, or outlined). The objective is to identify a "set" of three cards where each characteristic (symbol, color, number, and shading) either entirely matches or entirely differs across the set.

1.3 Game Setup and Rules

During the gameplay, twelve cards are placed on the table at a time. The player must identify a group of cards as quickly as possible within the allotted time. When a set of cards is correctly identified, they are removed and one point is accrued, and replaced by three new cards drawn from the deck. If the set of cards cannot be seen before the end of the allotted time, the computer wins and accumulates one point. The game continues until the deck is exhausted and no more sets can be formed. Scoring is based on the number of correctly identified sets, and the player with the most sets at the end wins.

1.4 Main challenge

The main challenges of playing the SET card game would be quickly identify patterns of different card attributes, managing competitive pressures from opposing players, adapting strategy to the dynamically changing state of the game, and maintaining sustained attention throughout the game. The first is due to the game's demand for quick recognition of sets based on color, number, shape, and shading within a time limit, where each attribute either matches all or differs in all three cards- a process that combines visual perception and cognitive speed. The second challenge stems from the real-time nature of the game, where players search for sets of cards at the same time, which creates a highly competitive environment that can put pressure on decision-making. The third challenge comes from the fact that the array of visible cards changes each time the game is played, which requires players to constantly adjust their strategy and scan for new patterns as the game evolves. Finally, staying focused is also critical, as the fast-paced and open-ended nature of the game requires players to pay constant attention to detail, which can be mentally taxing over the course of a long game. These challenges make SET a test not only of perceptual skills, but also of strategic thinking and mental endurance.

2| Algorithm

2.1 Introduction to the Algorithm

The SET card game implementation in Python, using the Pygame library, involves a blend of algorithms and data structures to manage game states, card properties, and interactions. The core components of the algorithm include card generation, set validation, and game state management, supported by an intuitive graphical user interface.
The main components of the algorithm include:
Card Initialization and Image Loading: Each card is initialized with specific attributes (number, symbol, color, shading) and an associated image.
Set Validation: Checking if three selected cards form a 'set' based on the game rules.
Game Loop: Handling events, updating game state, and rendering the game board.

2.2 Card Initialization and Image Loading

Data Structures Used:
Attributes in Card Class: Each card's properties (number, symbol, color, shading) are stored as integers. The image associated with each card is loaded and stored as a pygame.Surface.
Rectangles for Positioning: Pygame's Rect objects are used to manage the dimensions and positions of card images on the screen.

Pseudocode:

```
class Card:
    initialize(number, symbol, color, shading):
        self.number = number
        self.symbol = symbol
        self.color = color
        self.shading = shading
        self.image = load_image()
        self.rect = None
    load_image():
        filename = construct_filename_based_on_attributes()
        return pygame.image.load(filename)
```

Complexity:
Time complexity:
Setting integer properties (‘number’, ‘symbol, color’, ‘shading’) and a ‘None’ value (‘rect’) are all O(1) operations.
The ‘load_image()’ method call within the initializer also impacts the time complexity. Although loading an image involves disk I/O, which can vary in time based on the file system and other factors, for the purpose of algorithmic complexity, this can often be considered O(1) because it doesn't scale with the size of the input (number of cards).
However, it’s important to note that in practical terms, image loading can be a significant performance factor, especially if images are large or if I/O operations are slow. This isn't captured

by big-O notation which typically describes scaling with input size rather than absolute performance.
Space complexity:
Space Complexity: The space required to store four integer properties and a reference to a pygame.Surface object is constant, hence O(1).
The actual memory usage for the pygame.Surface object depends on the image size and color depth. While the reference to the surface is O(1), the image data itself can consume a significant amount of memory. This part is not captured when we say O(1) for the Card instance as we usually exclude external factors like image data in basic complexity analysis.

2.3 Set Validation

Data Structures Used:
Set for Attribute Comparison: Utilizes Python's set data structure to determine if attributes across cards are all the same or all different.

Pseudocode:

```
function is_set(card1, card2, card3):
    for each attribute in [number, symbol, color, shading]:
        create a set of values from the three cards for the attribute
        if size of set is 2:
            return False
    return True
```

Complexity:
Time Complexity:
The time complexity is O(1) because the operation involves a fixed number of iterations:
There are always exactly four attributes (‘number’, ‘symbol’, ‘color’, ‘shading’).
Each attribute check involves creating a set from three values (one from each card) and checking the size of this set.
Since both the number of attributes (4) and the number of cards considered (3) are constants, the total number of operations is constant, irrespective of other factors.
Space Complexity:
The space complexity is also O(1) for similar reasons:
A new set is created in each iteration of the loop for an attribute, but this set only ever contains up to three elements (since there are only three cards).
After checking one attribute, the set can be discarded or reused for the next attribute (depending on implementation), so the maximum number of additional space required at any time is for storing three distinct values and some minor overhead for the set structure.

2.4 Game Loop and Event Handling

Data Structures Used:

List for Cards on Table: A list to store the card objects that are currently visible on the table.
Set for Selected Indices: A set to track the indices of selected cards.

Pseudocode:

```
function main():
    while game is running:
        handle_events()
        update_game_state()
        render()
function handle_events():
    for each event in pygame.event.get():
        if event is quit:
            terminate game
        else if event is mouse click:
            update_selected_cards()
function update_game_state():
    if three cards are selected and form a set:
        remove_cards_from_table()
        replace_with_new_cards_if_available()
function render():
    display_all_cards()
    update_display()
```

Complexity:

Time Complexity: O(N) per frame

Handling Events ('handle_events'):

Iterating over events via 'pygame.event.get()' typically operates in O(m) time, where m is the number of events to process. In most frames, m should be small and constant, but it can vary based on user input.

Updating Game State ('update_game_state'):

The complexity here is determined by the operations required to check if the selected cards form a set and then update the game state accordingly. If the number of cards selected is constant (always checking exactly three), the operation to check for a set is O(1). However, removing cards and replacing them can depend on how many cards are on the table (N), especially if we need to shift cards in the list or refill the table.

Rendering ('render'):

Rendering each card is typically an O(1) operation, but since we need to render N cards each frame, the overall complexity becomes O(N). This includes drawing card visuals and updating the display.

Space Complexity: O(N)

The primary space consumption in your game is from:

Storing N card objects Cards on the Table

Set of Selected Indices: This set holds indices of selected cards, which in most gameplay scenarios will be very small (usually up to 3 for the SET game), but considering it in the overall context, it's still part of the O(N) space complexity due to the list of cards.

2.5 Conclusion

The SET card game implementation demonstrates efficient use of Python's capabilities in managing game states with minimal computational overhead. The chosen data structures are appropriate for the tasks, ensuring operations are performed in constant or linear time relative to the number of cards processed. This approach ensures that the game runs smoothly and is responsive to user input.

3| Implementation

The implementation of the SET card game in Python using the Pygame library focuses on an interactive and visually engaging user experience. This section will discuss the concrete implementation details, emphasizing the rationale behind the choices made regarding the program's structure, libraries, and specific functions.

3.1 Implementation Overview

The code is organized into a Python script that utilizes object-oriented programming principles. The Card class manages individual card properties and rendering, while the game logic, such as checking for sets and handling player interactions, is embedded in functions within the main game loop. Pygame is employed to manage graphical output and user interaction.

3.2 Detailed Implementation Discussion

Utilizing Pygame

Choice of Pygame:

Justification: Pygame is chosen for its simplicity and robust handling of graphical and event-driven programming in Python. It allows for easy rendering of images and handling of user inputs, which are essential for interactive games like SET.

Implementation:

Initialization and Setup: Pygame is initialized at the start of the main function, setting up the necessary environment for rendering and event handling.

Main Loop: The game loop continuously checks for events, updates the game state, and renders the current state to the screen.

Card Class Design

Choice of Attributes and Methods:

Justification: Each card's attributes (number, symbol, color, shading) directly correspond to the game's rules, which dictate that a set consists of three cards where each attribute is either all the same or all different across the cards. The load_image method simplifies the management of graphical representations, tying the logical state of a card to its visual appearance.

Implementation:
Image Loading: The load_image function constructs a filename based on the card's attributes and loads the corresponding image. This design allows for a dynamic loading of images based on card properties, reducing memory usage by loading images only as needed instead of preloading all possible combinations.

Game State Management
Data Structures:
Justification: Lists are used to manage the cards on the table due to their dynamic nature, allowing for easy addition and removal of elements. Sets are used to track selected indices because they provide efficient O(1) average-time complexity for add, remove, and membership test operations.
Implementation:
Event Handling: Events are processed in a loop, checking for quit events to close the game and mouse button events to update selected cards.
Set Checking and Card Replacement: After checking if three cards form a set, the cards are either removed or replaced, and the score is updated. This implementation ensures the game dynamically adjusts to player actions and maintains a fluid gameplay experience.

Rendering and Interaction
Graphical Output:
Justification: Effective visual feedback is crucial for user-friendly gaming experiences. Highlighting selected cards and displaying messages about game status (e.g., whether a set is valid or not) enhances user interaction.
Implementation:
Card Display: Cards are displayed using their stored image attributes and positions, updated based on user interactions.
Feedback Mechanisms: Visual cues, such as changing colors and displaying text messages, provide immediate feedback to player actions, which is key to maintaining engagement.

3.3 Conclusion

The implementation of the SET card game using Pygame in Python effectively combines object-oriented programming with event-driven design to create an engaging and responsive gaming experience. The choice of Pygame allows for straightforward graphical rendering and input handling, while the use of appropriate data structures and class design ensures that the game logic is both efficient and easy to manage. This approach not only adheres to the game's rules but also enhances playability through intuitive player interaction and visual feedback.

4| Manual

To facilitate the application and understanding of our developed algorithm, a comprehensive user manual has been prepared and made available online. We aim to ensure that even those with minimal background in this programming domain can effectively use and benefit from our software. To access the user manual, please visit our GitHub linked below.
https://github.com/Ruohann/SET/blob/main/README.md

4.1Prerequisites and Setup Instructions

Prerequisites

Python Installation: Ensure Python 3.x version is installed on your system. You can download it from the official Python website (https://www.python.org/downloads/).
Pygame Library: The game uses Pygame for its graphical interface To install Pygame, open your command line and enter: pip install pygame.
Operating System: The game should be compatible with Windows, macOS, and Linux operating systems that support Python and Pygame.

Setup Instructions

Download the Game Files: Obtain all necessary files, including Game.py and required image assets for the cards.
Game Images: Place the card images in a folder named images within the same directory as the Game.py. The naming convention for the images should match the pattern used in the Card class to load images.
Running the Game: Navigate to the directory containing Game.py and execute the following command in the terminal: ‘python Game.py’.

4.2 Usage Instructions

Starting the Game

Upon launching the game, the main menu will allow the player to select the difficulty. Options are typically 15s, 30s, or 60s, which may affect the number of cards dealt or the frequency of shuffling.

Gameplay

Selecting Cards: Use the mouse to click on cards to form a set. A set consists of three cards where each feature (number, symbol, color, shading) either all differ or are all the same across the three cards.
Validating Sets: After selecting three cards, the game will automatically check if they form a valid set. If so, they will be removed from the board, and new cards will be dealt if available.
Scoring: Players receive points for each correctly identified set. The exact scoring rules can be adjusted in the game settings.

Ending the Game

The game ends when no more sets are available or the deck is depleted. The final score will be displayed, and players can choose to restart the game or exit.

4.3 Code explanation

Card Class

The Card class is central to the game, representing each card's properties and behaviors:

```
class Card:
    """
    Represents a card in the game of SET with a number, symbol, color, and shading.

    Attributes:
        number (int): Number of shapes on the card.
        symbol (int): Shape type of the card.
        color (int): Color of the shapes.
        shading (int): Shading type of the shapes.
        image (pygame.Surface): Loaded image of the card for rendering.
        rect (pygame.Rect): The rectangle defining the card's position and dimensions.
    """

    def __init__(self, number, symbol, color, shading):
       # Initialize each card with provided attributes and load its image

        self.number = number
        self.symbol = symbol
        self.color = color
        self.shading = shading
        self.image = self.load_image()# Load the card image upon initialization
        self.rect = None # Will be set when the card is displayed on the screen.
```

Each card has attributes for number, symbol, color, and shading. The constructor (__init__) initializes these attributes and loads the corresponding image.

load_image(): This method constructs a filename from the card's attributes and loads the image for rendering.

Game Logic Functions

The game logic is encapsulated in several functions that handle the mechanics of the SET game:

is_set(card1, card2, card3)

This function checks if three cards make a set:

```
def is_set(card1, card2, card3):
    """Determines if three cards form a set based on their attributes.
    A set is formed if, for each attribute, the values are either all different or all the same.
    """

    for attr in ['number', 'symbol', 'color', 'shading']:
        values = {getattr(card1, attr), getattr(card2, attr), getattr(card3, attr)}
        if len(values) == 2:
            return False
    return True
```

It iterates over each attribute of the cards and uses a set to check if all attributes are either all the same or all different across the cards.

find_one_set(cards)

Finds the first valid set among the provided cards:

```
lef find_one_set(cards):
    """Finds and returns the indices of the first valid set found among the cards, or None if no set is found
    for i in range(len(cards)):
        for j in range(i + 1, len(cards)):
            for k in range(j + 1, len(cards)):
                if is_set(cards[i], cards[j], cards[k]):
                    return [i, j, k]
    return None
```

This function uses a triple-nested loop to try every combination of three cards and returns the indices of the cards that form a set.

Display and Interaction

Rendering cards and handling user interactions are crucial for game experience:

display_cards(screen, cards_on_table, selected_indices)

This method manages the display of cards on the game screen:

```python
def display_cards(screen, cards_on_table, selected_indices):
    """Displays the cards on the screen, highlighting selected cards.
    The function arranges cards in a grid layout and highlights cards that are currently selected.
    """
    card_width = 100
    card_height = 150
    gap = 20   # Spacing between cards
    start_x = (screen.get_width() - (4 * card_width + 3 * gap)) // 2
    start_y = 50  # Initial position for card display

    x, y = start_x, start_y
    for idx, card in enumerate(cards_on_table):
        image = card.image
        card.rect = pygame.Rect(x, y, card_width, card_height) # Define the position and size of the card
        screen.blit(image, (x, y))  # Display the card
        if idx in selected_indices:
            # Draw a yellow rectangle around the selected card to highlight it
            pygame.draw.rect(screen, (255, 255, 0), card.rect.move(0, 20), 5)
            pygame.draw.rect(screen, (0, 0, 255), card.rect.move(0, 20).inflate(10, 10), 5)
        x += card_width + gap
        if (idx + 1) % 4 == 0: # Move to next row after every 4 cards
            x = start_x
            y += card_height + gap
```

Cards are spaced and drawn on the screen. If a card is selected, it is highlighted by drawing a red rectangle around it.

5| Conclusion& Discussion

5.1 Findings

In this report, we detailed the design, algorithm, and implementation of the SET card game in Python using Pygame. The game architecture leverages object-oriented programming to encapsulate card properties and game logic. Key data structures such as lists and sets efficiently manage cards on the table and track selected indices. The computational complexity is optimized to O(N) per frame, ensuring responsive gameplay even as the number of displayed cards changes. The game has been successfully implemented, tested, and is available for download on GitHub, demonstrating practical application of programming concepts and effective use of Python in interactive game development.

5.2 Strengths

One of the game's primary strengths is its simplicity in rules coupled with deep strategic complexity. It enhances cognitive skills like quick-thinking and pattern recognition, making it not only entertaining but also educational. Additionally, the game is universally accessible, requiring

minimal setup and being easy to learn, yet it presents a challenging endeavor for players to master due to its open-ended nature.

5.3 Shortcomings

However, the SET game is not without its shortcomings. The high cognitive load required can be daunting for new players, potentially leading to frustration. This barrier to entry might deter some from fully engaging with the game or enjoying it. Moreover, the game's simplicity in design might not appeal to players looking for thematic or narrative depth.

5.4 Improvements

These focus on optimizing the current gameplay and making the game more accessible and enjoyable for all skill levels.

Digital Integration:

Tutorials and Adaptive Difficulty: Introduce tutorials to help new players understand the game quickly. Adaptive difficulty levels can cater to players of various skill levels, ensuring the game remains challenging and engaging.

Analytics and Feedback: Utilize data analytics to provide players with insights on their gameplay, offering feedback on their strategies and helping them improve.

Optimizations and Handling Edge Cases:

Caching Mechanisms: Implement caching strategies to minimize redundant computations, thus speeding up gameplay, especially in a digital version.

Concurrency in Multiplayer Scenarios: Develop robust handling of multiple player interactions to ensure smooth and fair gameplay when claims on sets are made simultaneously.

5.5 Extensions

These are new features and elements that can be added to expand the game’s scope and its appeal to a broader audience.

Enhanced Social Features:

Clubs and Regular Meet-Ups: Promote the creation of SET clubs and organize regular meet-ups through digital platforms or local venues, enhancing the community aspect of the game.

Social Tournaments: Facilitate regular online and local tournaments to foster a competitive environment and keep the community engaged.

Advanced Technological Features:

Online Multiplayer and Competitions: Enable global competitions through an online multiplayer mode, allowing players from different locations to compete.

Virtual Reality (VR) Implementations: Explore VR technologies to offer a more immersive and interactive experience, making the game more engaging and visually appealing.

7| References

Wikipedia contributors. (2024b, June 19). *Set (card game)*. Wikipedia.

https://en.wikipedia.org/wiki/Set_(card_game)#