Sets Project Report:

Team Members: Mujie Wang, Xiaorong Wang, Dylan Soemitro

Summary:

For the final project, our group wrote the code for SET game in python, using some of the data structures we learned and CS thinking we developed during the semester, and by using the Pygame[1] package that we learned on our own.

Background

Based on SET official website, set is American's most popular card game. Each deck contains 81 unique cards.

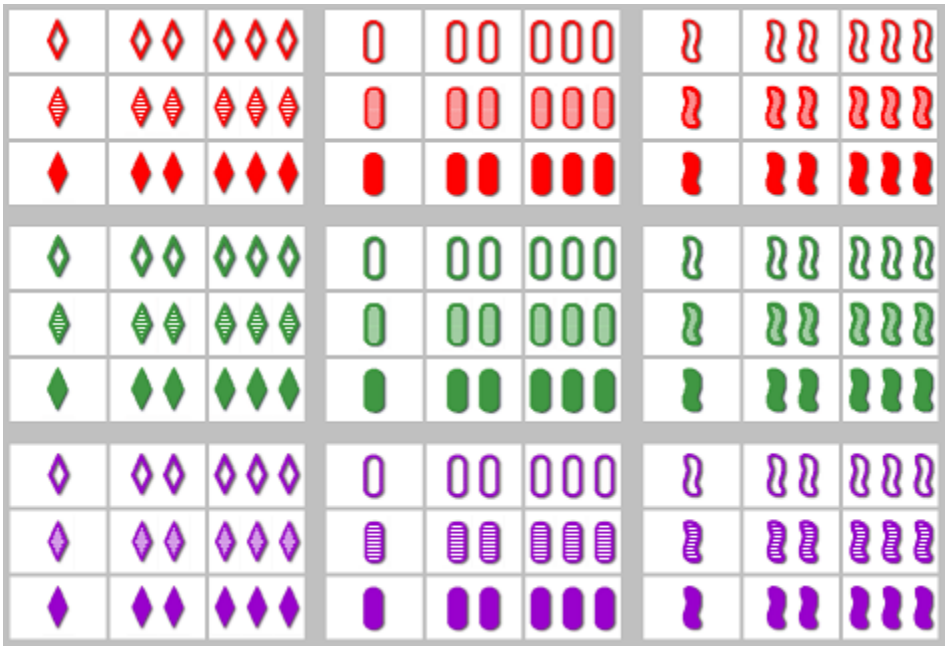

Figure 1[2]: All 81 unique cards

Each card has four features: number, color, shape and shade. Each feature has 3 possibilities, detailed in the table below.

[1] https://www.pygame.org/news
[2] https://thecardgamepiseigo.blogspot.com/2017/02/set-card-game.html

Number	Color	Shape	Shade
1	Red	Diamond	Empty
2	Green	Oval	Half
3	Purple	Squiggle	Full

Table 1: Features of cards

A set of cards is a group of three cards such that, within each feature, they either are all different or all the same. In our implementation, the game starts with a cardboard of 16 or 20 random cards from the deck, depending on whether the player chooses the easy or hard mode (easy = 20 cards, hard = 16 cards). The player then is able to select three cards. The program will verify if the three cards make a set. If so, the program will replace the old cards with three random new cards that have not been used yet. If there is no set on the board, the program will automatically fill in 3 cards, until there is a set on the board. When stuck, the player can click on “hint” button. The program will highlight one card that appears in the most amount of sets on the board.

Motivation:

(Mujie)

Motivation 1: There is a deck of SET cards at math lounge. Sometimes after class, a bunch of friends will sit around and play it together. There are two people, whose name I will not mention, are very good at this game. Whenever I play against them, they can find a set within

2 seconds, which makes the game not fun for me. So, I decided to practice on my own.

Motivation 2:	Recently we lost one card in the deck, which make the game not playable anymore. Losing one card has such a great impact. In Python, we can never lose a card. So, we decided to implement the game in Python.

Contributions of each team member:

Team member	Contribution
Mujie Wang	• **Responsible for motivation behind the project** • **Worked on logic for the game itself** • **Created methods for the background code** • **Debugged background code**
Xiaorong Wang	• **Created methods for the background code** • **Created introduction screen, ending screen, hint button by using Pygame**
Dylan Soemitro	• **Learning Pygame and implementing the graphics of the program**

- **Linking the back-end code with the front-end**
- **Used Photoshop to slice image to get individual PNGs of cards**

Methods, data structures, packages, programs and algorithms used:

Background:

1. **Find the third card:**

Recall that in order to be a set, three cards, for each feature, need to be all the same or all different. “All the same” is easier to implement, while “all different” is a bit trickier. We took advantage of the cycle structure of Z mod 4 and modified it a little bit. First, we label the choices of each feature with integers 1, 2, 3. Observe that if we add two together, it will give the third number or 0, which we redirected it to 2.

The structure of this game is a graph, as shown in Figure 2. Each card is a node. There is an edge connecting each pair of nodes. And each edge can be represented as the third card.

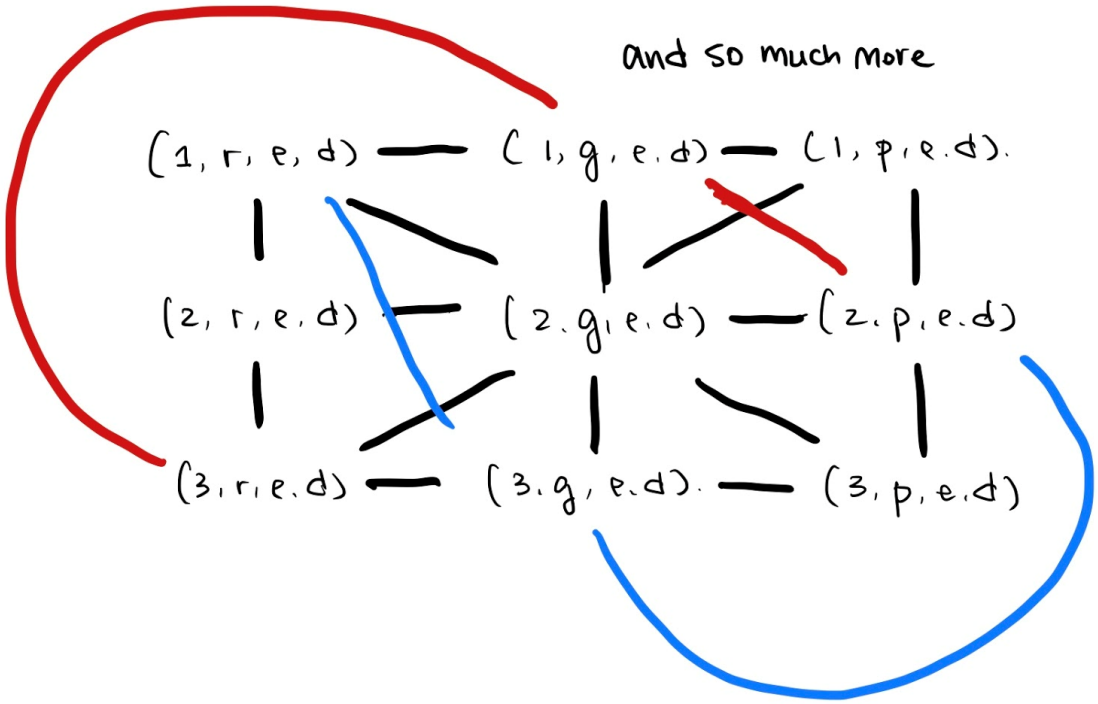

Figure 2: Graph of cards

2. Encode a 4-tuple into an integer.

Since each card has four features and each feature has 3 choices, we label each card uniquely with a 4-tuple with entries 1, 2, or 3. For the fairness and fun of the game, we want the cards shuffled, but it is harder to shuffle tuples than to shuffle integers. So, we use the idea of ternary to encode the tuple into an integer in decimal. So if we have a tuple (a, b, c, d), we first delete 1 from each entry and turn it into (a*, b*, c*, d*), and the corresponding integer will be a* x 3^3 + b* x 3^2 + c* x 3 + d*. We can also reverse the procedure to convert an integer from 0 to 80 uniquely to a card.

3. Using stack to store shuffled card.

After a deck is shuffled (81 random integers without replacement), we put the shuffled cards into a stack and pop the first one when needed.

4. **Packages used**

Package	Use
Pygame	Used mainly to create the user interface of the game. It is the base of all other implementations.
time[3]	Random package is used to shuffle the stack of 81 cards each time the program starts. This is to make sure each time when the user plays the game, the cardboard given is completely random and will change in different games.
random[4]	The random package helps us achieve the goal to resemble a real-life board game scenario where cards pop up in completely random orders.

Front end:

1. **Pygame**

Used to make the GUI of our project. Pygame was used due to its simplicity and ease of use in making games with basic graphics. We used Pygame to make a window where the game would be played and implement buttons which could be clicked using the press of the mouse. This window also was able to display the actual cards (stored in a folder as PNGs) instead of the tuple representing the card (as discussed above). Having buttons where the user just has to click them instead of having to keyboard input to the terminal, as well as being able to display the actual cards, greatly improves the user experience in playing the game.

[3] https://docs.python.org/3/library/time.html
[4] https://docs.python.org/3/library/random.html

2. Adobe Photoshop[5]

As there was only one image containing all the cards (Figure 1_ we had to use photoshop to slice the image into 81 PNGs to be used in the game.

Challenges:

Challenge	**Reasoning**
Using Photoshop	Because there were no individual images of the cards, we had to slice each individual image from a larger image. This process was time-consuming and difficult as Photoshop started lagging as I sliced more and more images.
Using Pygame	As we were unfamiliar with Pygame, it took a while learning and getting used to it.
Connecting the codes	We initially worked separately, which proved to be difficult when connecting the front end to the back end. The way we programmed our individual solutions were different, meaning that it was troublesome to understand and use each other's codes. We eventually started working together much more closely to

[5] adobe.com/products/photoshopfamily.html

create a team project and not a combination of individual ones.

Future improvements:

Proposed Improvement	Explanation
Score Calculation	For example, we can include the number of hints we used. Later, we can also include the number of cards left when there is no set in the deck left.
High score	We can save the user's last score and high score for reference
Two Player Mode	We can make a two-player mode where two players need to claim "set" before they choose cards, and see which player finds more sets. This mode can be offline or online.

Lessons Learned:

Lesson	Explanation
Working closely together	As explained earlier, for clarity and consistency it would be better to work closely together as a team
Logistics	Find out better times where everyone can work on the project together in person