SET!

The family game of visual perception.

By: Arnout Dam (4388054), Alec Drost (4993179) and Isa van Woudenberg (2183234)

Date: 02/2025

Git repository: https://github.com/Isavw1905/SET.git

Table of contents

1.1 Background and Questions

Background of the Problem

SET is a card game based on pattern recognition. The goal is to find a "SET" of three cards that follow specific rules. Each card has four attributes: number, shape, color and shading, with three possible values for each.

A valid SET consists of three cards where, for each attribute, the values are either all the same or all different. Mathematically, a SET card can be represented as a list with four values in {1,2,3}. A SET is a group of three lists where each attribute follows the SET rules.

In this project, we developed a Python program where a player competes against a computer in SET. There are always 12 cards on the table. The first to find a SET scores a point. If the player does not find a SET within a time limit, the computer gets a point and immediately reveals the SET.

Challenges

Structuring the code and learning to use Pygame

One of our first challenges is figuring out how to structure the game using classes and functions. We also need to decide how to create classes for the cards, manage the deck, and handle user input.
Eventually, we need to visualize the game, for which we will need to use Pygame. None of us have any experience with Pygame. Therefore, this will impose a slight challenge for us.

Creating the opponent, the computer

Another challenge we will encounter is the designing of a code in python that finds SET's instantly. The computer must be able to quickly scan the twelve cards and detect a valid SET. However, it should not reveal it right away. Instead, if the player doesn't find a SET in time, the computer should display the SET and score a point. Keeping track of the score and showing it clearly on the screen is another element of the game we need to keep in mind.

Representing cards and checking for SET's

The cards are the most important part of the game. To make sure the game works properly, we need to create a class for the cards that stores their four properties. Within the class we will also need to define a function that checks if three chosen cards form a valid SET. To keep the game fair, we must shuffle and draw cards from the deck in a way that prevents predictable patterns.

Designing the game interface

The game needs to be easy to play and understand. We must decide how to show the twelve cards clearly, how the player will select cards, and how to give feedback when a SET is correct or incorrect. With Pygame, we need to figure out how to highlight selected cards and update the game screen smoothly.

Implementing buttons and player interaction

To make the game easy to play, we need buttons for selecting cards and submitting a SET. We must also add visual effects to show when a button is selected or deselected.

Optimizing the game for speed

Since the game has to check many card combinations, we need to make sure our code runs efficiently. The goal is to make the game smooth and quick. Understanding how much time and memory our SET-checking algorithm uses will help us improve performance.

By solving these challenges step by step, we aim to create a well-structured and fun SET game in Python that gives players an exciting challenge against the computer.

1.2 Algorithm

Before implementing the game, we first needed to decide how to store and process the cards. We chose to represent each card as a list of four integers, with each integer corresponding to one of the card's four properties: colour, shape, shading, and number. Additionally, each card has a string name that matches the filename of its image. Both representations are stored together in a nested list, making calculations and image loading straightforward. The properties are defined as follows:

Properties:	Colour:	Shape:	Shading:	Number:
1	Green	Diamond	Empty	One
2	Purple	Oval	Filled	Two
3	Red	Squiggle	Shaded	Three

Table 1.2.1: properties of cards and corresponding numbers

With this structure in place, we began developing the Card class in Python.

Classes

The class for the cards starts with the initialization. The Card class is responsible for storing a card's properties, including colour, symbol, shading, and number. To make image loading easier, we created a string conversion function that formats the properties into a structured name.

One of the most critical functions in this class is check_set(), which determines whether three given cards form a valid SET. The function works by verifying that, for each property, the three cards either share the same value or all have different values. Instead of checking directly for a SET, our approach is to check whether the given cards fail to be a SET. If any property does not follow the SET rule, the function immediately returns False and breaks. If all properties pass the test, the function returns True.

Since this function always evaluates exactly four properties, it runs in $\mathcal{O}(1)$ time complexity. However, this only works efficiently because we assume the input always consists of exactly three cards. If we were to generalize the function for a variable number of cards (n), the complexity would increase to $\mathcal{O}(n)$ due to the additional loop iterations. In addition to checking SET's, we implemented the pick_set() function, which searches for a valid SET among twelve displayed cards.

The pick_set() function works by generating all possible combinations of three cards and passing each combination into check_set(). If a valid SET is found, the function returns the three cards, and the loop breaks. If no SET is found, it returns False. Since the function iterates through three

loops (one for each card in a combination), its worst-case time complexity is $\mathcal{O}(n^3)$. However, since the maximum number of displayed cards is twelve, this complexity remains manageable.

Lastly, we needed a way to randomly select and remove cards from the deck. The random() function returns a random card. It also removes the card elements from the corresponding input lists.

This is how we defined the class for the cards, it can pick and check a SET and it can also randomly make cards. We wanted the game to work with buttons, therefore we needed another class. Because we had no previous experience with Pygame, we looked on the internet on how to make a button class and found a video, we adapted the code from the video to make it work in our game. The video is listed with our other sources. This class takes an image, position and the scale to create the button, if the button is clicked on screen, the draw() function returns 'True'.

Game-loop

The game begins by generating two lists of 81 unique cards: one containing the numerical representation of each card and another containing their string names for image loading. The twelve cards displayed on the screen are selected randomly using the random() function from the Card class, ensuring that each card has both a list-based representation and a corresponding string name.

During the player's turn, they can select three cards. Once three cards are chosen, the check_set() function determines whether they form a valid SET. If they do, the player earns a point, and the selected cards are replaced with three new ones. However, if the player submits an invalid SET or if the in-game timer runs out, the turn passes to the computer.

When it is the computer's turn, the pick_set() function searches for a valid SET among the displayed cards. If a SET is found, those cards are replaced, the computer scores a point, and the timer resets. If no SET is found, the game replaces the last three cards with new ones and resets the timer. The game continues until the deck is empty, and no valid SET's remain on the board. At this point, the game ends, and the player with the most points wins.

This explanation covers the core game loop. Additional features such as the main menu, difficulty selection, pause/help button, exit button, and visual indicators for selected cards contribute to the user experience. These elements primarily rely on if-statements and event handling in Pygame, which are well-documented in the code with comments.

1.3 Manual

Getting the program running

To play the game, the player must unzip the zip file in our git repository. All of the documents should now be within 1 folder, within this folder there should be one folder with pictures of all the cards called 'kaarten'. Once the documents are downloaded correctly the player should make sure pygame is installed. If they're unsure because they have never used it before they can follow the steps explained on: https://www.geeksforgeeks.org/how-to-install-pygame-in-windows/. After pygame has been installed, the file called 'Main code.py' can be opened on any preferred IDE. As soon as the player runs the code the game screen should open and the game can begin.

The start screen

Once the game has been opened the start screen will be displayed with a total of six buttons, as seen in (Figure 1). The player can click on each of these buttons. The four coloured ones describe the difficulty levels, if any of these are clicked the game will start on the selected difficulty. In the top left there's a button with which a player can immediately exit the game. In the top right corner players can find the help button, which will be displayed throughout the entire game. This will display a new screen and pause any timer while in-game.

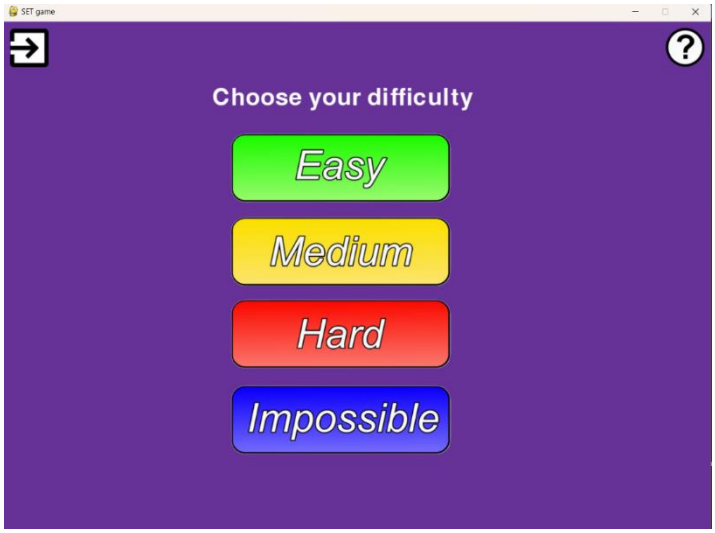

Figure 1 The start screen

The help screen

If the question mark in the top right corner gets clicked on, the help screen will be displayed (Figure 2). This screen will show the player the rules of the game and give three options; returning to the game, returning to the start screen, and closing the game. These options are all buttons once more, so the player simply has the click on any of the three buttons to close the help screen and go back to any screen they want.

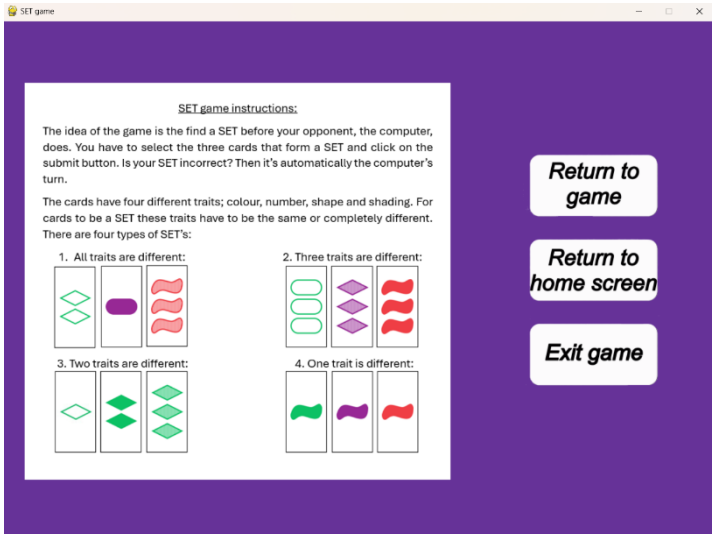

Figure 2 The help screen

The game screen

Once a difficulty has been chosen the game screen will appear (Figure 3). The idea of the game is to collect as many SET's as possible within the time limit. To do so the cards that make a SET have to be selected, this can be done by clicking on the cards. A selected card will have a yellow edge around it, like the card in the top left in Figure 3. The player can select a total of three cards, if the player wants to deselect a card it has to be clicked on again and the yellow outline will disappear.

Once a player has found a SET and selected all three cards, they have to click on the submit button. The computer will then check if the selected cards are indeed a SET. If it is the player will get a point. Was the player wrong, then it's the computer's turn. Did the timer run out before the player was able to submit a SET, then it's also the computer's turn.

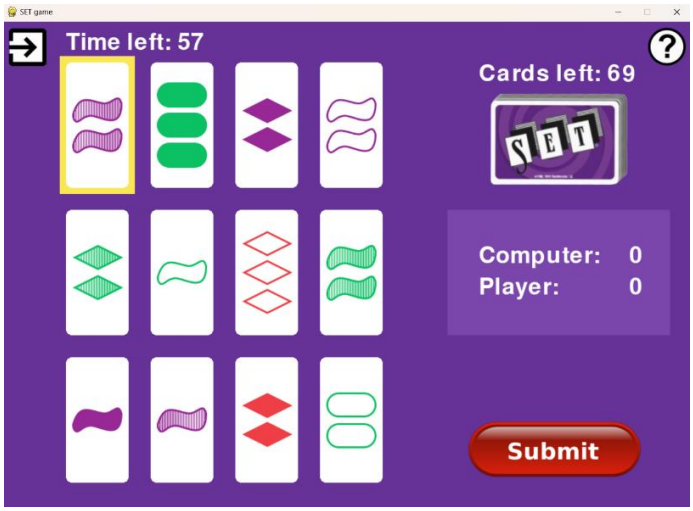

Figure 3 The game screen

If it's the computer's turn the computer will check if there is a valid SET within the cards on the table. If there is the cards will get a blue outline (Figure 4) for three seconds so the player can see what they missed. The cards will then be removed and new cards will take their place. If there is no SET within the displayed cards the computer will remove the three cards in the right column and replace them. The timer will then start over and the player gets another turn.

The game continues for as long as there are cards within the stack or SET's to find. If neither of these are the case then the game will display an end screen depending on the final result. When on that screen the player can push any button on their keyboard to return to the start screen and either start a new game or close the game.

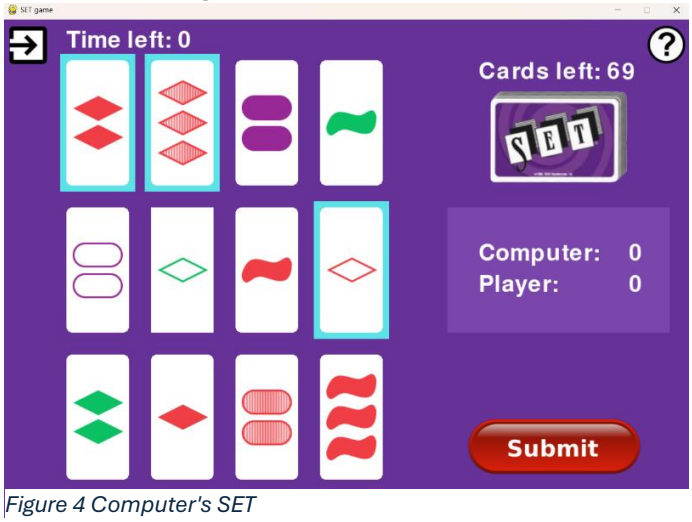

Figure 4 Computer's SET

Adapting the code

The game can be adapted, however most adaptations would be quite useless. One adaptation that could be useful however would be changing the length of the timer for the different difficulties, depending on the player's preferences. The duration of the timers can be found in line 350, 355, 360 and 365, depending on the difficulty (Figure 5). Here you can change the numer to the preferred duration in seconds, times 1000, plus 900.

```
347
348             #decide difficulty and start game
349             if easy_button.draw(screen):
350                 timer = 60900
351                 start_time = pygame.time.get_ticks()
352                 begin_screen = False
353                 game_screen = True
354             elif medium_button.draw(screen):
355                 timer = 30900
356                 start_time = pygame.time.get_ticks()
357                 begin_screen = False
358                 game_screen = True
359             elif hard_button.draw(screen):
360                 timer = 15900
361                 start_time = pygame.time.get_ticks()
362                 begin_screen = False
363                 game_screen = True
364             elif impossible_button.draw(screen):
365                 timer = 5900
366                 start_time = pygame.time.get_ticks()
367                 begin_screen = False
368                 game_screen = True
369             #both corner buttons
370             if instructions_button.draw(screen):
371                 help_not_needed = False
372                 paused_time = pygame.time.get_ticks()
373             if exit_button.draw(screen):
374                 running = False
375
```

Figure 5 Determining the duration of the timer

1.4 Conclusion / Discussion

We started the report with explaining the game SET, it's a game about finding the most SET's the fastest. With our program, the computer functions as the player's opponent. If the player does not find a SET within a given time, the computer takes a SET and gets a point. We explained our game-loop and algorithms in paragraph 1.2, our game-loop mostly works on if-statements and event handling, these parts in our code are explained using comments. Our class for the cards does most of the calculation work, it can check if three given cards form a SET and find a SET from any amount of cards. It is also able to randomise the card making process. We also made a class for the buttons, because we wanted the game to work with buttons instead of an input bar. In paragraph 1.3 we give a manual on how to play the game and it explains what each button does. Finally we will discus points of improvement and possible extensions on the game.

Improvements

Our code is very long and in some places quite repetitive, we could improve our code by shortening it. Our way of showing the cards on screen and checking if the cards are being clicked takes up 115 lines of code, we could probably implement this in one of our classes or make some kind of for-loop. This would save a lot of the space. The same goes for how we load our images; there is a way to implement this in our card class, we just don't know how. The same goes for a few more repetitive parts, if we designed the card class to preform these operations instead of putting the same code for all twelve cards, then this would greatly decrease the size of our code. We haven't been able to achieve this, which is mostly due to our unfamiliarity with classes. Because of our lack of previous experience with programming, we chose to use if-statements, which we were familiar with, rather than using classes.

Another place for improvement is our algorithms, the function that finds a SET from twelve cards has a time complexity of $\mathcal{O}(n^3)$, this is not very efficient. The code makes every possible combination of three cards to find a SET. We could also work with the fact that for every two cards there only is one card to make it a SET. Then we would only need to make all combinations of two cards and check if the third card is part of the remaining ten. This would give a time complexity of $\mathcal{O}(n^2)$, which is more efficient than what we have now.

Extensions

While playtesting our game we have already come up with and added a few extensions. For example; the computer shows which cards it chose, therefore you can see which SET you missed. We also added buttons to make the game more engaging. There is a help button that pauses the game, a button with which you can exit the game at any time, and a button to go back to the begin screen and select a new difficulty. Additionally we added an end screen showing the final scores. Possible new extensions could be adding animations when the cards get replaced to make the game even more engaging. We could also add sounds and music for the same purpose. However, we think our game mechanics are complete and do not require extensions. The only extensions would be cosmetic and not change the game-play in any way.

Reflexion

We ran into quite a few challenges while working on the project. Whenever we reached a particularly challenging part of the project, we did research online over the weekend. The next tuesday we would discuss our solutions and additions to the code together so we were all aware of each other's work. This way everyone was involved with the code at all times, which made it easy to work together. We were able to check any changes in the code easily because we worked together in a git repository. Because of our smooth cooperation the game was running quite quickly, which gave us the opportunity to playtest and add any extensions we were personally missing in the game. Overall, we faced some difficulties surrounding the code, but because of our collective effort we never got stuck on them for very long.

Sources

Coding With Russ. (2021, May 26). *PyGame Beginner tutorial in Python - Adding buttons* [Video]. YouTube. https://www.youtube.com/watch?v=G8MYGDf_9ho