FINAL EXAM - The Game of Set

Instructions

- This final exam is due **Monday, May 5 BEFORE SCHOOL STARTS**. I will be checking the timestamps on Schoology to verify the time at which your project was submitted.
- Your grade will be based on whether the game works. You cannot score more than 10 out of 20 if it doesn't work.
- **There will be no extensions unless there is a doctor's note. Do not bother asking. You have a full week to work on this.**
- If you work with your partner, you both have to turn in the assignment separately.
- Do not copy code from the internet, as that is plagiarism.
- If you choose to try to use AI, make sure you cite your sources as best you can. Please note that AI is great, but your code needs to work in Processing with the file as given and directions as given. Making sure that your choice of words clearly tells the AI what you need can be challenging.
- You may not talk to other humans about this project with one exception: the teacher.
- Don't change the graphics implementation!
- Grading will be: 20 points for project completion, **You are not allowed to work on the final during class.**
- Don't panic! It's only a final.

Introduction and Background

From Wikipedia's page on SET, https://en.wikipedia.org/wiki/Set_(game):

"***Set*** is a real-time card game designed by Marsha Falco in 1974 and published by Set Enterprises in 1991. The deck consists of 81 cards varying in four features: number (one, two, or three); symbol (diamond, squiggle, oval); shading (solid, striped, or open); and color (red, green, or purple).[1] Each possible combination of features (e.g., a card with three striped green diamonds) appears precisely once in the deck."

"A set consists of three cards satisfying *all* of these conditions:

- They all have the same number or have three different numbers.
- They all have the same symbol or have three different symbols.
- They all have the same shading or have three different shadings.
- They all have the same color or have three different colors.

The rules of *Set* are summarized by: If you can sort a group of three cards into "two of ____ and one of ____", then it is not a set."

You can play SET at this web site: http://smart-games.org/en/set/start

Spend some time playing the game so you get a feel for how it works.

Download the latest version of Processing if you don’t have v4.0 or later on your computer. Classroom computers have it already.

NOTE: Processing won't work unless you have Java installed on your machine. This shouldn't be an issue in most cases since you should have downloaded the Java SDK a long time ago to set up Eclipse. However, if you acquired a new machine and haven't put Java on it yet, you will need to do so to run Processing.

If you do not have your own computer at home, please see me so I can arrange a loaner for you.

If you are using your own device, download the SET Starter Kit

To open the SET starter kit:

1. Find the ZIP file that you downloaded from the above link and double-click on it to produce the SET_Final_Official2025 folder.
2. Inside the folder, find the file SET_Final_Official2025.pde and open it. **YOU MUST OPEN THE SET_Final_Official2025 file specifically. If you open one of the other files in the folder, you won’t be able to run your code because the first tab is the entry point for a Processing program.** You may need to right click and do "Open With" and select Processing. I've seen .pde files be opened as text files and you really want Processing. If you see the code in Processing, you should be good to go.

Before writing any code...

It is helpful to read the code in the different tabs to get a sense of how things will work. Don't stress if you feel you don't understand everything at first. Things should become more clear as you write the methods needed to solve this project.

General guidelines: When you write code...

Use meaningful variable names. If something goes wrong, it will make it easier to diagnose your own problems. If you need to ask me for help at some point, it will make it much easier for me to understand your code. For example, when you write nested for loops to iterate on blocks in the grid, please use col and row as index variable names instead of i and j. An index variable named i has its place, but if you have information about the context, something meaningful makes life so much nicer for the reader. Note that the indices on the grid are of the form (column, row), not the other way around.

Thinking about writing the SET game

The only files that you are allowed to change on this test are:

- **Grid.pde**
- **Set_Game_Logic.pde**
- **Timer_Procedures.pde**
- **Utility_Functions.pde**

TREAT THE OTHER FILES AS LIBRARY FILES THAT OTHER PROGRAMMERS NEED. POINTS WILL BE TAKEN OFF IF YOU CHANGE THEM! IF YOU WERE TO CHANGE THEM, YOU WOULD BREAK THEIR PROGRAMS!

The code is written such that you should not need to change Utility_Functions.pde, but if you decide to write helper procedures, please put them there. It's important to solve the problems, so there may be times when you have a solution approach that isn't what we were thinking about when writing the test. Helper procedures should be in one of the above four files.

Note that the .pde files correspond to the tab names that are in the program. The tabs will typically be in **BOLD** type when referenced in these directions. Variables and methods will typically be in monotype. (Not sure if font types matter, but maybe they will be a little bit helpful.)

Note the enum in the **SET_Final_Official2025** tab (the first tab). It is used to create symbols without worrying about values that help to describe the state of the program. The different states are:

- State.PLAYING: Normal play mode where the user is considering which cards may form a set
- State.EVAL_SET: The user has selected three cards to be a possible set; now the program evaluates that input and responds accordingly
- State.FIND_SET: The user cannot see a set and has asked for a set to be identified by the program
- State.GAME_OVER: What it says
- State.GAME_PAUSED: The Pause button has been used and the cards are to be hidden until the game is resumed

Take a look throughout the program to make sense of how these symbols are used in the code.

At the end of **SET_Final_Official2025** tab, there is a method called showMessage(). Please make note of it. **You must NEVER change it**, but the value in the message variable will determine what gets displayed in the message area of the display. showMessage() uses a special form called switch, with a structure that looks a bit like cond in Scheme. However, it is different in a critical way: each line of a cond has its own predicate. With switch, the input variable has a value and the line of code that is performed depends on that value.

Directions for writing the SET game

NOTE 1: It is critical not to skip any steps. If you do things in order, life should be fine. If you miss a step anywhere, you may find yourself going back a long way to get things right.

NOTE 2: Do not try to run the program until you have completed all nine steps of #1, below. If you do, you will get a loading error and will probably have to force quit the program.

1. In the **SET_Final** tab, write the method newGame() on line 193 which should:
    1. Create a new deck.
    2. Create a new grid.
    3. Set the score to zero.
    4. Set the variable currentCols, which contains the current number of columns in play, to 4.
    5. Change the state to State.PLAYING.
    6. Set the message variable to the number which will result in the user seeing "Welcome to SET!".
    7. Make the grid add (currentCols * ROWS) cards to the board. **You will need a for loop and you will do this by adding a card that is freshly taken from the deck using the deck's deal() method inside that for loop.** (You won't actually see any cards on the board quite yet; that will change when the addCardToBoard() method is written later in Problem 6.)

        Make sure that you increase currentCols when you add a column to the grid!

    8. Set timeElapsed to 0 since no time has elapsed when a new game is started.
    9. Set runningTimerStart to millis().

        If you have done all nine steps correctly, the program should show a gray background with a timer running at the top.

        NOTE: If you get stuck, take a look at newTestGame() that is just underneath.

2. In the **Set_Game_Logic** tab, write new methods allSame() and allDifferent(). These methods each take three ints as inputs and return true if the inputs are all identical and all different, respectively.

    For #3 - #6, division by 3 and/or modulus 3 (the remainder operator %) are your friends.

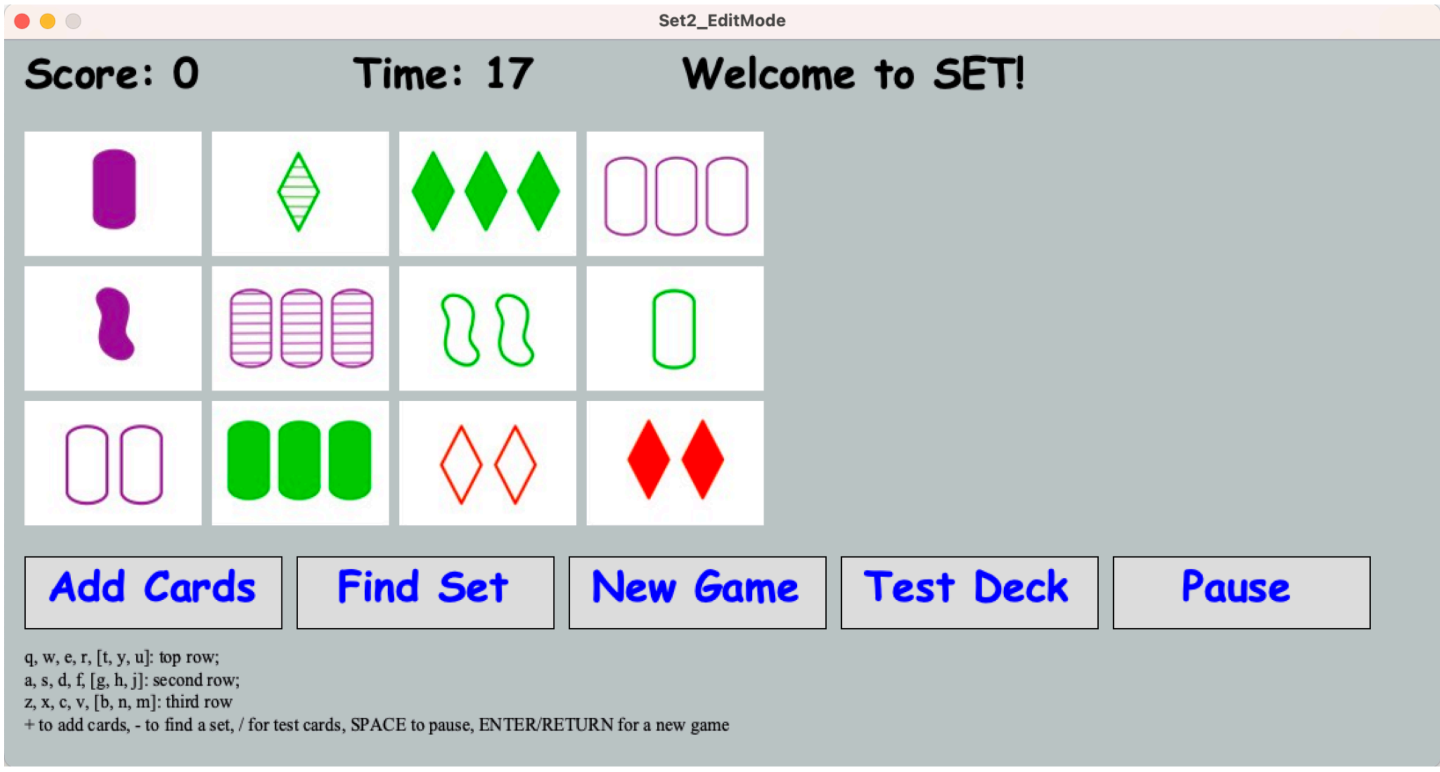
3. Fill in the methods sameColor(), sameShape(), sameFill(), and sameCount() in the **Set_Game_Logic** tab. These methods return true if the three cards have the same property and false otherwise. Use allEqual(). **You will want to refer to the sprite sheet set-cards.jpg, which is in the project folder you downloaded, to see how cards are organized by row. The commonalities amongst cards based on their properties will hopefully become evident from inspecting that sheet.**

4. Fill in the methods diffColor(), diffShape(), diffFill(), and diffCount() in the **Set_Game_Logic** tab. These methods return true if none of the three cards share the same property and false otherwise. Use allDifferent().

5. In the **Set_Game_Logic** tab, write the method isSet(). It returns true if the three cards form a set, and false otherwise. It can be done with a one line return statement if you use logical AND (&&) and logical OR (||) properly.

6. Write addCardToBoard(Card card) in the **Grid** class. It should use the variable cardsInPlay to figure out which column and row to use. **To do this, you will need to figure out which column and row the next card should be added to and then assign the card to board[col][row].** It should add one to cardsInPlay after a card is added to the board. It should also use the ROWS constant to ensure the correct number of rows.

   At this point, you can highlight cards and see a lot of the Set functionality. This is a good time to verify that the logic you wrote for problems 2--5 is correct and to fix things if something doesn't quite work. It will look something like this:



q, w, e, r, [t, y, u]: top row;
a, s, d, f, [g, h, j]: second row;
z, x, c, v, [b, n, m]: third row
+ to add cards, - to find a set, / for test cards, SPACE to pause, ENTER/RETURN for a new game

   If that isn't the kind of thing you are seeing, then probably something went wrong in problem 6.

7. Write the **Grid** method addColumn() as follows:
   1. If there are no more cards in the deck, change the message number so this will be indicated and back out of the method. `return` with no inputs is how to break out of a void method.
   2. If there are no sets on the board, then add five points to the player's score and add three new cards to the board. Change the message number to indicate that cards have been added to the board. Use the `addCardToBoard` method appropriately.
   3. If there is a set on the board, subtract five points from the player's score and change the message number to indicate that there is a set on the board.

8. **(This might be the most challenging problem of this project. We would recommend that you think things through carefully, draw diagrams, and do whatever you need to do to sort out the logic before trying to write code.)**

   In the **Grid** class, fill in the method `removeSet()`. The precondition for removeSet() is that a set has been found and the ArrayList `selectedLocs` contains the *locations* of the cards in the set that must be removed. (see Location class in the **Location** tab to see how locations work. Write the following code:

   1. If the number of cards in play is greater than 12, or if there are no cards left in the deck, then new cards will not need to be dealt. However, the board will need to be consolidated so that the number of columns is decreased by one and the number of cards is decreased by 3. The way to do this in the most game-friendly way is to move cards from last locations on the board into the locations in selectedLocs. There is a little bit of code here to help you get started. You will need to be careful about how you are using selectedLocs and the board array.

      You can set up a situation where there are more than 12 cards without a set by clicking the Test Set button. You may find it helpful for testing this situation.

   2. If there are exactly 12 cards in play and the deck is not empty, deal three new cards into the locations on the grid specified in the selectedLocs array.

9. In the GAME PROCEDURES section of the **Grid** class, write the boolean method `gameOver()` which returns true if the game is over. Think carefully about what it means for the game to be over.

10. Fill in the method `timerScore()` in the **Timer_Procedures** tab. If the player finishes in under five minutes (300 seconds), then one point should be added to the score per second under five minutes. There is no penalty to the score if the player finishes in over five minutes. Remember that the timer values are in milliseconds, not seconds and the score should be considering full seconds. Also note that Processing has a max() procedure that can return an int; use it.

11. In the method `processTriple()` in the **Grid** class, fill in the code for what should happen when the game ends:
    1. Update the variable state appropriately.
    2. Capture the time of the end of the game in the variable `runningTimerEnd` (use the `millis()` method).
    3. Update the final score by calling `timerScore()`.
    4. Change the value of the message variable so the fact the game is over is made clear to the player.

If you have written all of these correctly, then the game should work and you're all SET!

- **If your game doesn't run you will not score more than 10 out of 20. Do not procrastinate on this project as it's really hard to get work done if you put that much time pressure on yourself!**

- **If you haven’t done so already, share your contact information with your partner!**