Formative assessment - Part 1

INTRODUCTION TO PROGRAMMING

DEPARTMENT OF ELECTRONIC ENGINEERING, UNIVERSITY OF YORK

1 Brief

The increasing popularity of video games during the COVID-19 period has prompted companies to create new and interesting games. One such company CompX has decided to create a game to boost mental performance and memory by creating a simple yet effective brain training single-player video game that players can play on their PC anytime with proven memory benefits. They have come up with a working title for this game "Card-Match".

You have just been employed by CompX to design, implement and ship the game. CompX has provided a set of technical specifications leaving enough room for creative exploration to engage the player better.

For now your task is to create a program which displays playing cards, face down in a 7x7 grid. If a user clicks on a face down card, it should be revealed for 2 seconds. After that the card is hidden again and, another card can be picked.

The specifications are as follows:

1. The 48 cards will consist of the values from the Ace (1) till the Queen, of all the suits. The **kings will not be included in the set of cards**. Specifically, the set shall consist of Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen; for all four suits i.e. Hearts, Diamonds, Spades and Clubs. A total of 48 cards.
2. As soon as the game is started, a window must appear showing a set of cards placed facing down i.e. the cards must not show their values or their suits. The cards must be randomly placed, and there should be no duplicates.
3. The game will be played on a 7 by 7 grid. The central grid space will be empty thus removing the need for one card, leaving enough grid space for 48 cards
4. With the initial setup of face-down cards, clicking on any card will reveal the card. However, there can only be one revealed card on the grid at a time. Thus, after clicking on a card, a countdown timer is started. When the timer reaches zero, all the cards are reset to face down and the player is allowed to click again.
5. Specifically, clicking on a card will start a timer which will hide the revealed card after 2 seconds. Thus, once revealed, a card will be hiding again after 2 seconds and the player will be allowed to click on another card to reveal it.

6. **The game would also need to keep a record of the sequence of cards that have been clicked on, including their position on the grid, their value and suit.**
7. There should be a **debugging mode** which will display all cards faced-up when activated.

The game logic (i.e. matching suit, card number etc.) is going to be released next week. The developer needs to focus on displaying the cards on the graphics window and implement the card interaction; i.e. clicking the card will give the suit and number in console (Python shell).

Their internal designer produced a set of images, each representing a single card (including the kings) and provided them to you in a .zip file, along with two choices as the back of the cards. They have shared the **.zip file on the Wiki** along with their rendition of what the game should look like with all cards facing up and all cards facing up.

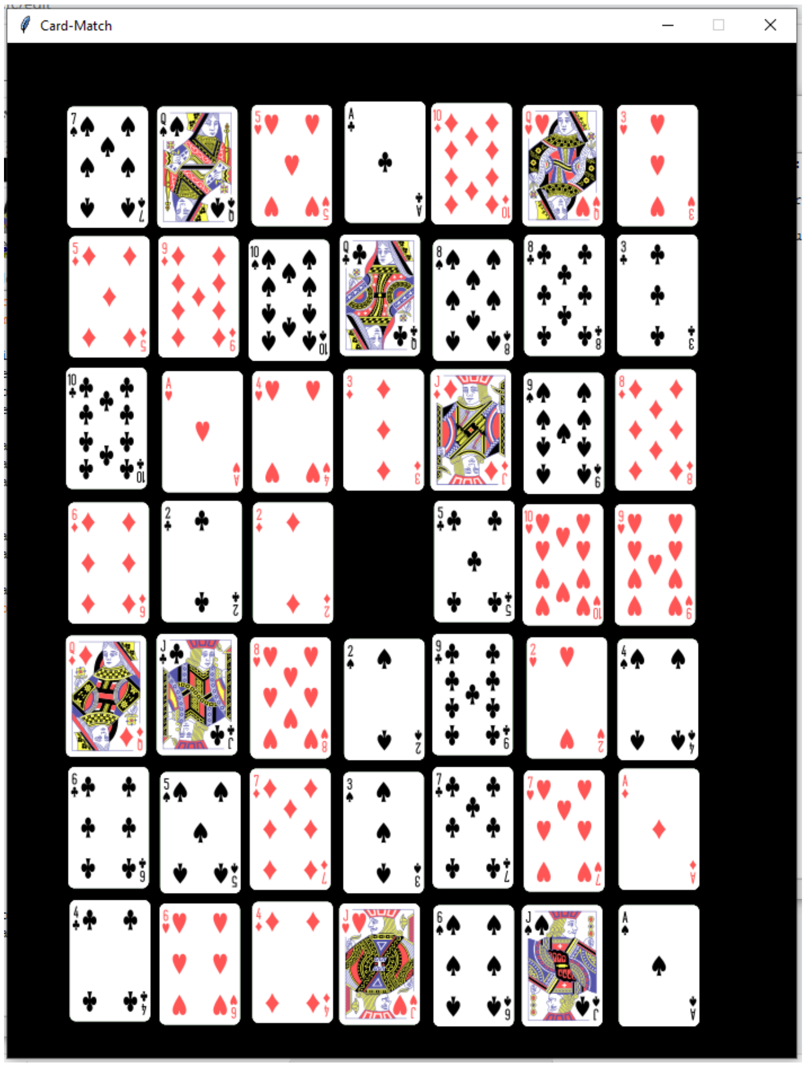

Figure 1: All cards facing up.

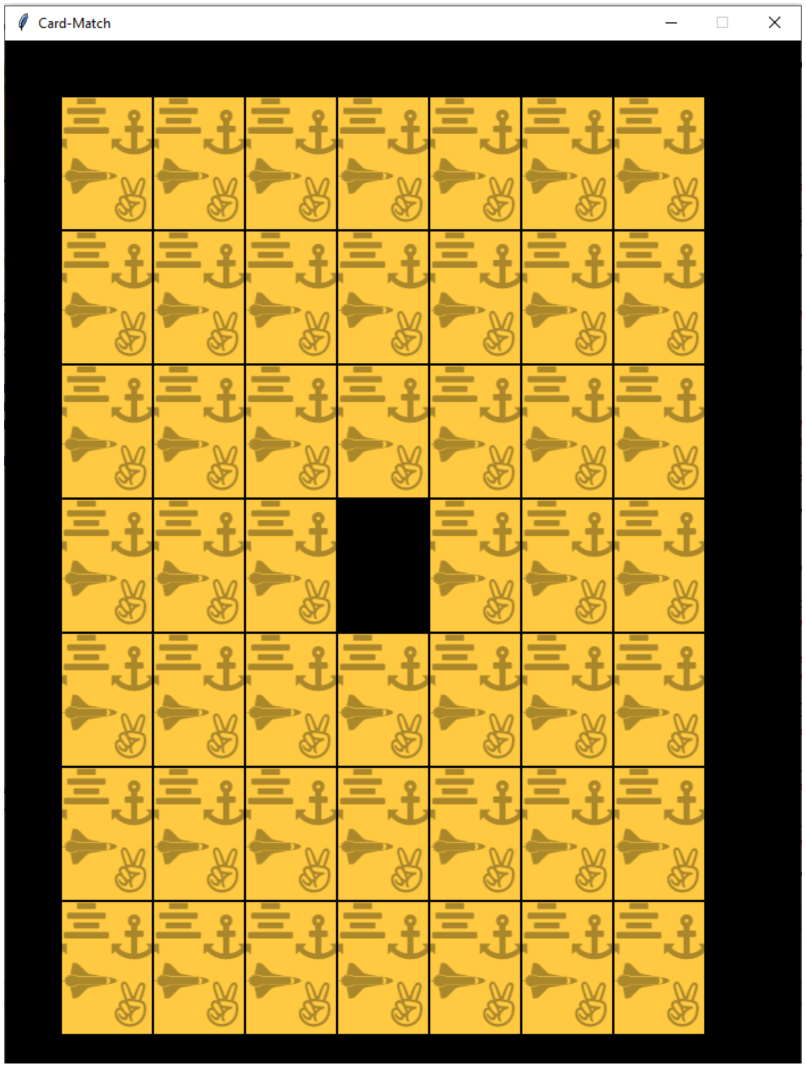

Figure 2: All cards facing down.

You should use the york_graphics.py library provided to make the vision of this videogame come to life. Additionally, the program needs to adhere to the specifications provided, while being flexible enough to implement the logic to be provided by them in a week's time.

2 Guidelines

This assessment can be logically divided into 3 parts

1. **Frontend:** The look and feel of it. This involves the creation of a grid and drawing the card images on the grid. The drawing of cards can be done either face-up or face-down depending on the interactions with the mouse.
2. **Interaction:** The only way of interacting with the game is using the computer mouse. Fortunately there's functions in the `york_graphics` library that can figure out when and where a mouse was clicked. The returned mouse coordinates can be used to determine which card might have been clicked on.
3. **Backend:** This is the game logic which shall be released next week. The game logic will be a variation of card-matching techniques. For example, the game could involve clicking on similar value of cards, or similar suit of cards.

It is important to remember to plan the code out, before writing anything. The use of flow-charts and pseudo code can greatly streamline the process of implementing code and prevent unnecessary frustration.

This week you should aim to:

- Create a plan of implementation using flowcharts (and pseudo code if you prefer)
  - Make sure you plan in advanced to accommodate for the game logic that is going to be revealed next week
- Use the `york_graphics` library and functions within it, to build the whole game
- Use the `drawImage(`*`filename`*`)` function to draw the card images at a specific location on the graphics window
  - The card images are provided as a .zip file containing .png files on the module wiki page. This function can work with .gif as well as .png files.
  - The card image file names are named according to the index of the card in a standard sorted card deck. Please ensure necessary logic to prevent drawing the Kings.
  - The .zip also contains images for representing the back side pattern of the card.
  - Note the image resolution (width x height) of the card images to determine where to draw them on the canvas window.
- The implementation is up to the programmer, but below are some useful tips:
  - Use `list( range( ))` to create a list of card numbers (see How to convert a range to a list in Python for reference)
  - Use random to "shuffle ( )" the cards randomly (see Python Random shuffle() Method for a quick syntax reference)
  - Use list indexing to determine which card to draw next on the grid
  - Use appropriate conditional logic to not draw in the centre of the grid

- Use `waitForMouseClick()` function to determine where the player clicked on the screen and use appropriate logic to figure which card the player clicked on.
- Once a valid click is detected, redraw the canvas to show the card that was clicked on
- Start a timer by calling the "sleep()" function for 2 seconds. Refer to section 13.2 in Lab03 script for details
- Once 2 seconds has passed, refresh the canvas by hiding all cards again
- Keep a record of which card was clicked on, their value and suit (e.g. the player clicked on the 23rd card, which was the 3 of hearts)
- Remember that the whole sequence of clicking on the cards needs to recorded; i.e. you need to keep track of the cards that have been clicked through another list.