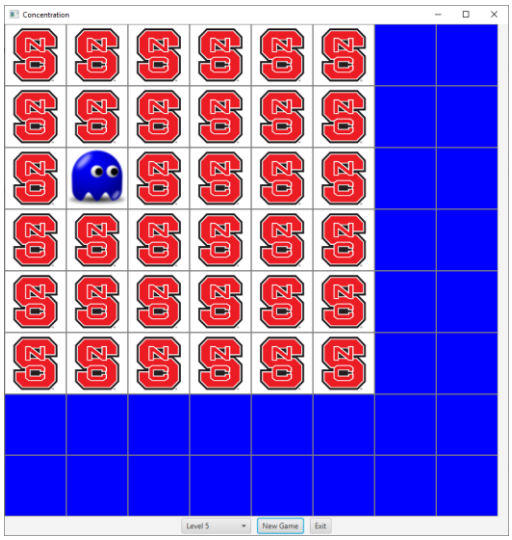
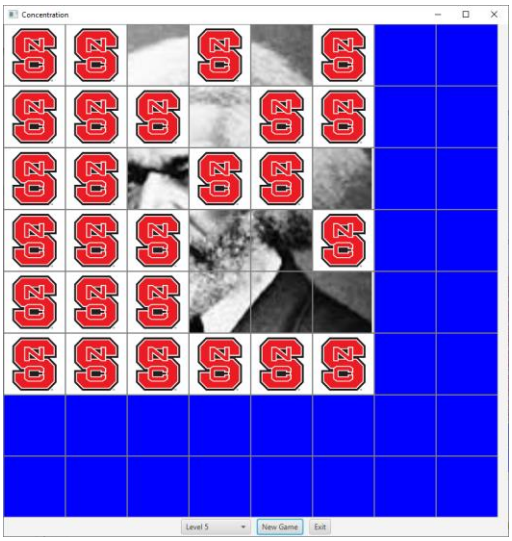
COMP167 Major Program 3 – Spring 2025

You will implement the matching game named Concentration for this programming assignment. The user will be presented with an NxM grid of cards with a symbol on the back. The player can only have 2 cards flipped over at once. The goal is to flip over 2 cards with the same symbol (symbols are placed in the game in pairs). Once all the matches have been found, the game ends.





Card Class:

The Card class will be an extended StackPane class that will simulate a single card. The card has the following properties:

- *flipped* (boolean) which indicates whether the symbol is displayed (true) or the back of the card is displayed (false).
- The *matched* property (boolean) will indicate if the card has been matched.
- The property *path* (String) will contain the file path of the image.
- The *image* (Image) will store the actual image object.
- The image is displayed using an ImageView object.
- *row, col* (int) indicate the row and column of this card in the grid NxM. These values are used if you want to show a new big image as the cards are being matched instead of black for each card. So, these are optional.
- *numRows, numCols* (int) The number of rows and columns used in the **current game** (determined by the selected level). This is used if you want to show a big new image as the cards are being matched instead of black for each card. So, these are optional.

All the properties are private and will have appropriate getter and setter methods. There should be a no-arg constructor and a constructor that is passed the path of the image. You are free to design your own images, use the ones I have posted or download them from the internet. **Make sure that your images are rated "G"**. My suggested size for an image is 64x64 pixels. Here are methods that you will implement for card:

- flipCard() – Flip the card to the other side. Use the StackPane clear() method to remove the displayed image from the card.
- setCardAndImageSize(int width, int height) – Set the size dimensions of the Card and the ImageView object in the card.
- setPath(String path) – set the path property, populate the Image object using the path and populate the ImageView object using the Image object.
- setMatched() – change the card properties such that it is considered matched.
- setGridPos(int r, int c) set the values of the optional row, col values described above.
- setGridSize(int nr, int nc) set the values of optional numRows and numCols values described above.

CardGridPane Class:

The CardGridPane class will contain an 8x8 grid of card objects (Can you guess what layout it should used?). It will have the following properties:

- cards (2d array of Card objects)
- cardList (ArrayList<String>) The pathnames of all the available card images.
- MAXROWS (int) the number of GridPane rows (8)
- MAXCOLS (int) the number of GridPane columns (8).
- currentRows, currentCols (int) the actual number of rows and columns used in the current game as set by the user selected level.
- cardSize(int) the sidelength in pixels of the square card.
-

The no-arg constructor for the class should create cards with width and height of 64 and add them to the CardGridPane. A second constructor passes the card size as the single parameter. You should also assign each created Card object to the corresponding location in the 2D array (e.g. cards[i][j] = new Card();). The cards 2d array will give you convenient access to cards once they are added to the StackPane. Other methods that you will need to implement are:

- void setCardImages() – this method will assign the image path for each Card (using the ArrayList of Card paths) that will be used in the current game using the setPath method of the Card class. **Note:** the setPath setter method for the Card class should not only assign the value to the path property of the Card but should also read the image file and assign it to the Image object. The image object should then be stored in the ImageView object property of the Card class. You do not call this method until the user has selected a level which in turn determines the number of rows and cols of the CardGridPane that will be used. This method can be used internally within the CardGridPane class to reassign the images to the Cards after they have been rearranged for a new game.
- void shuffleImages() – Shuffle the image paths in the ArrayList.
- Card getCard(int r, int c) – return the card object referenced by location [r][c] in the cards 2d array.

- void initCards(int rows, int cols) – After the user selects the level, use this method to set game up. You have to disable all cards not used in the current game, create the ArrayList of image paths, shuffle the ArrayList and then call the setCardImages() method described above.
- You will also need setters and getters for the rows and cols properties.
- void createCardImageList(int size) – Create an ArrayList of image paths with a length of size. The ArrayList is already a property of the class.

GamePane Class (extended BorderPane):

The GamePane class will contain a CardGridPane Object, a StatusPane that will hold the game status information (e.g. the number turns and the game timer). A turn is defined as 2 clicks. The class will also have a CommandPane with controls for setting the level, starting a new game and exiting the application. The game level determines the actual number of Cards used for the game (level 1 – 2x3, level 2 – 2x4, level 3 – 4x4, level 4 – 4x6, level 5 – 6x6, level 6 – 8x8 (feel free to add other levels that are less than 8x8 in size). The controls that appear in the command and status panes should be declared as properties of the GamePane class so that they will be accessible by the listener methods that will be called to handle your events. For example, the Button to exit the application should be declared as a data field/property of GamePane. Besides the Panes and controls mentioned above, this class will probably need at least the following additional properties:

- rows, cols (int) – the number of actual rows and cols of cards needed for the game as specified by the chosen level (i.e. these values are not set until the user selects a level).
- numClicks (int) – tracks the number of mouse clicks during each turn. When this value hits 2, you should check for a match.
- numMatched (int) – The number of cards matched. When this value hits (rows*col/2) then the game is over.
- clickedCardOne, clickedCardTwo (Card) – References to the two cards that were most recently clicked.

The constructor for the GamePane class is responsible for instantiating and adding the other Panes. Create two constructors: a no-arg constructor that assumes the card size is 100 and a one argument constructor that has the card size as an argument. I declared my GamePane as an extended BorderPane in my solution with the CardGridPane object in the center region. Other methods:

- newGame() – Make changes to all the appropriate properties to start a new game.
- registerCardListeners() – register MouseClicked (or MousePressed) listener with each card.

Image Files:

Once you have downloaded or created your 64x64 pixel images, name them image_0.png (or .jpg depending on the type of image) through image_31.png and save them in a subdirectory named *images*. You should use images that all have the same image types (all jpg or png). Eventually, you must load the pathnames into an ArrayList. Each file will be included twice in that ArrrayList since the game uses image pairs.

```
int counter = 0;

 //add image paths to ArrayList
for (int i = 0; i < (rows * cols)/2; i++) {
    images.add("images/image_" + counter + ".png");
    images.add("images/image_" + counter + ".png");
    counter++;
}
```

Note: I have listed a minimal set of properties and methods. You can add additional properties and methods as needed.

Grading:

You should keep each level separated so that you can turn in a complete level if you do not get to the last level. No student should receive a zero on this assignment.

Level 0 (10pts) – Program Design

- Complete UML Class diagrams for the Card, CardGridPane and GamePane classes. Base these diagrams off the complete Level 5 solution. The diagrams should be added to your Level 0 branch.

Level 1 (40pts) – Create a GamePane that just contains a CardGridPane and the command pane. The command panel should contain a functional Exit button. For this level, your CardGridPane should show an 8x8 grid of Cards with each card flipped (i.e. all images should be visible). You will need to load the image file names in the images ArrayList and then call the setCardImages() method in the CardGridPane class. You should also set the flipped property of each card equal to true.

Level 2 (50pts) – Change the Level 1 problem so that each Card is initially unflipped. Add a MouseListener to each Card that will change the flipped property of the Card to true and issue a flipImage() for the Card so that it displays the symbol/image. From my experience, use the mousePressed event instead of mouseClicked.

Level 3 (65pts) – Modify your Level 2 solution so that the command pane contains either a group of radio buttons or a combo box to select the game level. The Listener for the level control should set the rows and cols value for the CardGridPane using the setter methods. The other Cards that are outside of the grid for the current game should be disabled. I designed my application so that it would display all disabled panels with a gray square. So, at the beginning of each game I disable all 8x8 cards. When the user chooses a level, I then enable all Cards within the grid size for the game.

Level 4 (85pts) – Modify your Level 3 solution so that it uses the AnimationTimer class. Your GamePane class should include a property that counts user mouse clicks. In the mousePressed method mentioned in Level 2, check the value of the clickCount variable. If it is 0, then just increment it and flip the card. If it is 1, then that means that this is the second click. You should start the timer and include a conditional

in the handle() method that code executed after 800 milliseconds (0.8 seconds) have expired. The code under the conditional in the handle() method should:

- Check the paths of each of the clicked cards to see if they match. If so, set the matched property of both cards to true and change the color each card to black indicating that they have been matched. If they do not match, just set the flipped property for each card to false and flip both cards so that their images are no longer visible.
- The clickCount should also be returned to 0.
- If there is a match increment the property in the CardGridPane that tracks the number of matches so that you will know when the game is over.

You should also define two variables in your GamePane class that will reference the Cards that are clicked by the user. This will enable you to know which cards were clicked when checking for a match and when you must flip them back over when there is no match.

Note: The ArrayList can be shuffled using the Collections.shuffle() method.

Level 5 (100pts) – Add some bells and whistles for your game.

- First, add a Label that displays and updates the number of turns that it takes to complete a game. When a game is complete, a message should pop up indicating the game is over.
- Add a new game button that will start a new game **at the current level.**
- Add code to your AnimationTimer handle() method that plays a sound for a match and a different sound for a non-match. Research the AudioClip class for playing short sound clips.

Extra Credit (Choose only one):

- Use a configuration file the keeps track of the best game score for each game level. When the game ends, a Dialog (new modal Stage) should pop up with the score and the date the score was achieved for that level (or you could just display the information for all levels). The Dialog should also have a reset button to reset all top scores (15pts).
- Add a game timer to your application. As soon as the user clicks the first card the timer starts. On screen, the timer is updated every tenth of a second. The timer stops when the final match is made. Create a new extended Pane class for this feature. The actual timer numbers are drawn on the panel canvas using a graphics Text object. Do some research and find a nice digital looking font. Of course, this class will use the AnimationTimer from Level 4. The class will also need public methods that start the timer, stop the timer, reset the timer to zero, and 3 methods that return the current hour, minute and second (15pts).