AP ACTIVITY **4.1.2**

Concentration (AP)

INTRODUCTION

How many different types of game boards have you played with? How many of those could be represented as a 2D array? In this activity, you will create one of those games, a game commonly called Concentration.

Materials

- Computer with BlueJ IDE

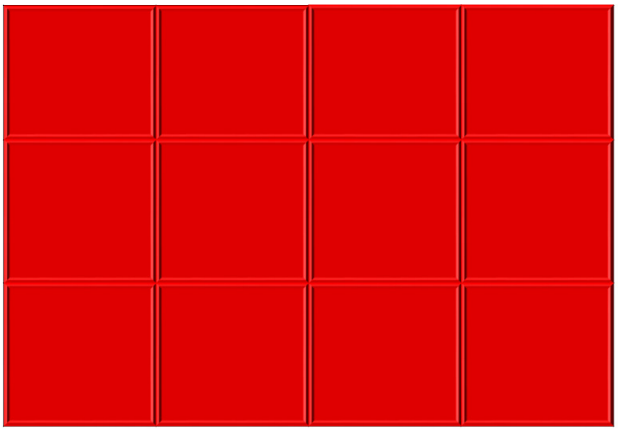
RESOURCES

Free Response Question: GrayScale
Resources available online

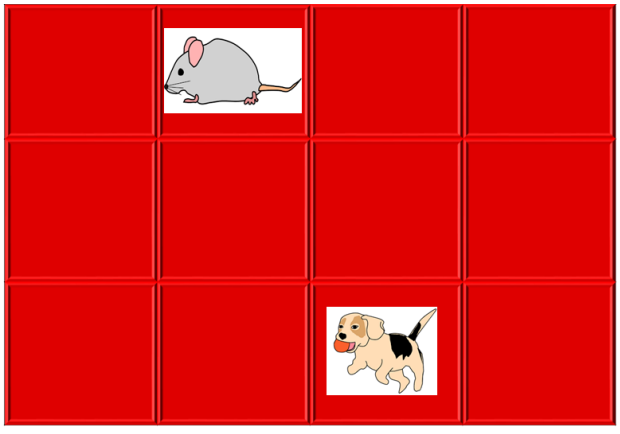
Procedure

Part I: How to Play

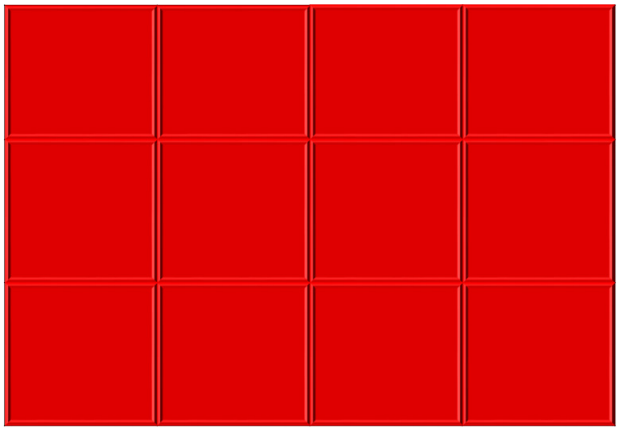
Concentration is a game of matching pairs of cards. Cards start face down on the board, and then players turn cards face up, two at a time, trying to make matches.



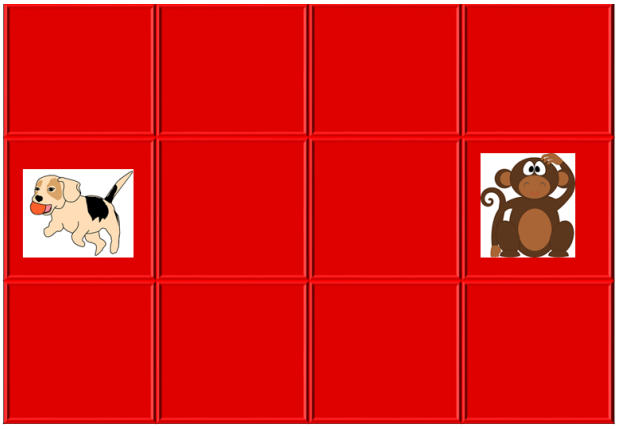
The game of Concentration starts with a set of cards or tiles face down.



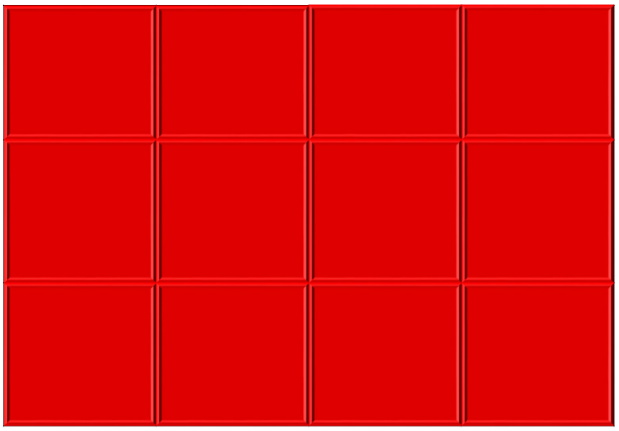
A player turns over 2 tiles to try to match the hidden cards. If the tiles do not match, the tiles are turned face down again.



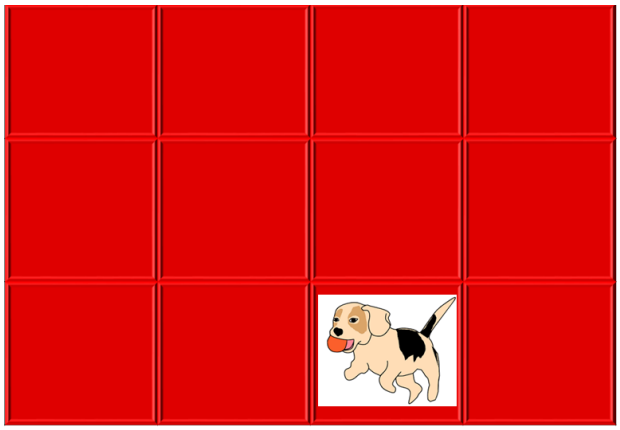
The player (or a new player) takes a new turn and selects a set of tiles to turn over.



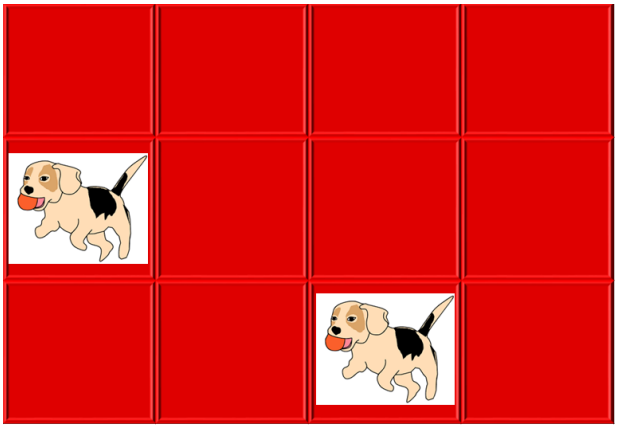
Here, the new tiles don't match, but the player remembers seeing the dog before!



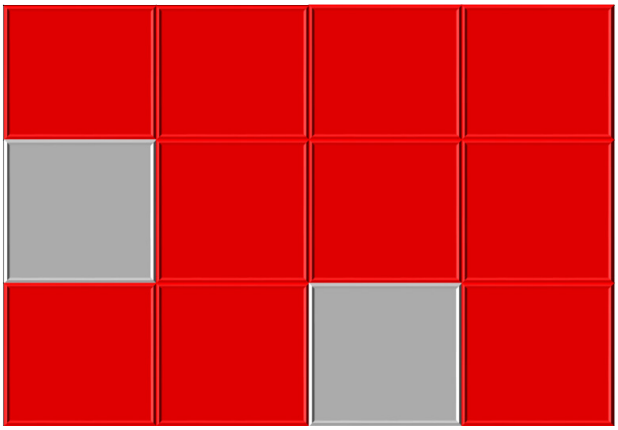
The tiles are turned face down to end the turn. Now, the player tries to find the dog tile from a previous turn.



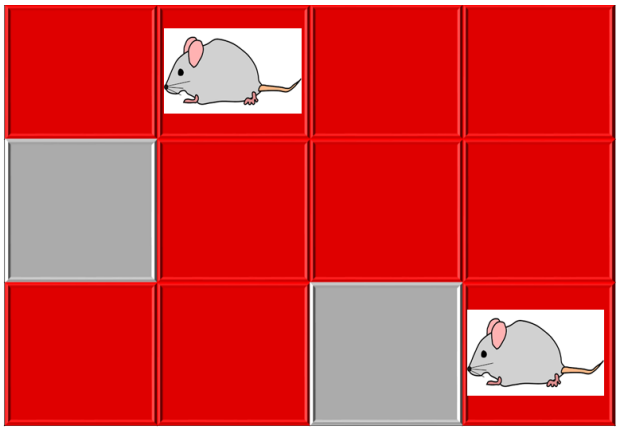
The player finds it,



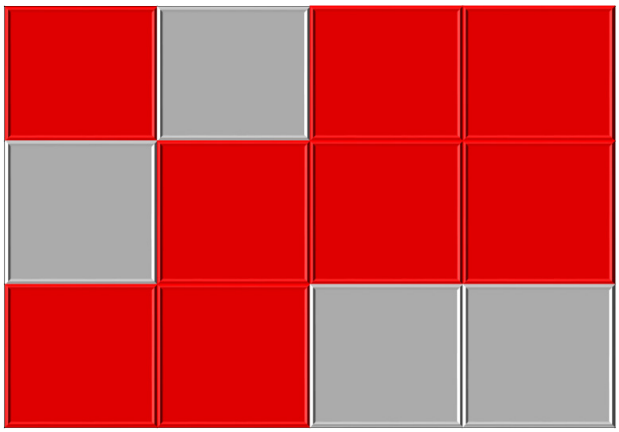
and turns over the other dog tile to make a match!



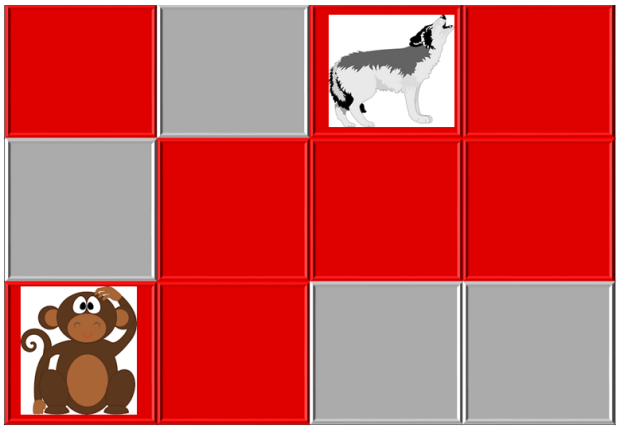
Matching tiles are removed from the game and play continues.



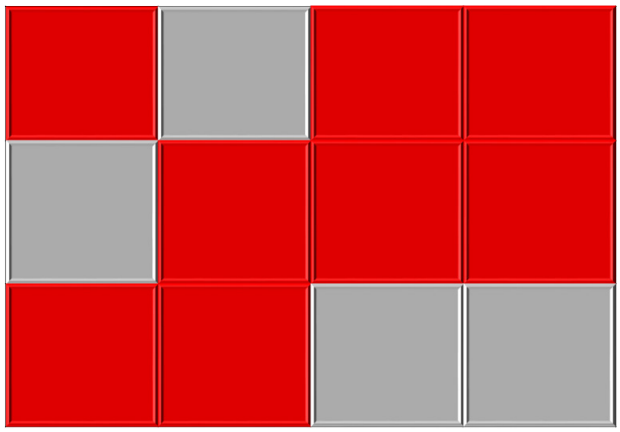
On a new turn, another match is made,



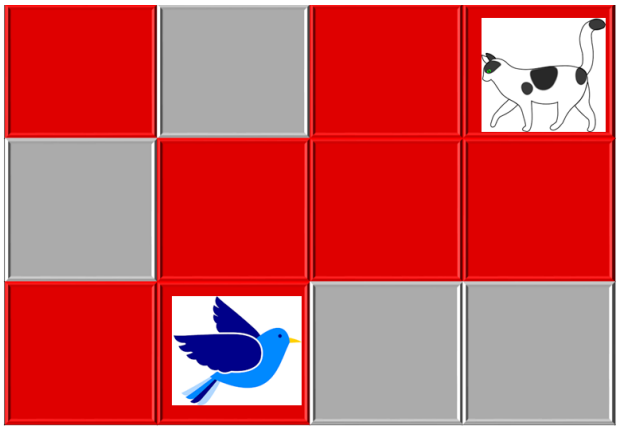
and the cards are removed from play.



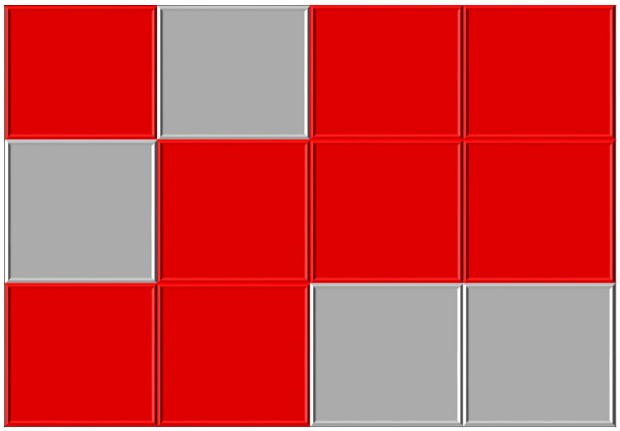
Play continues, these tiles don't match,



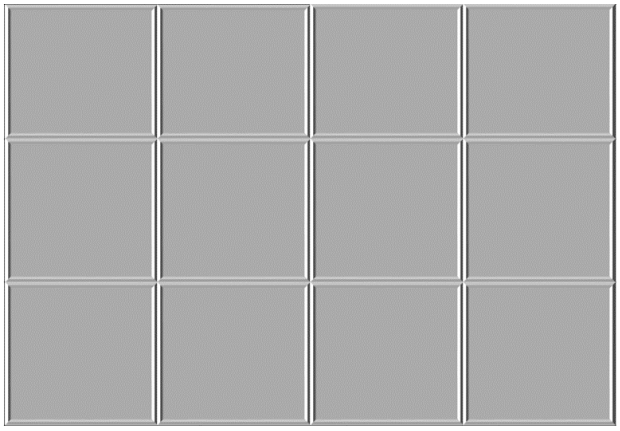
so the cards go face down again.



Play continues,



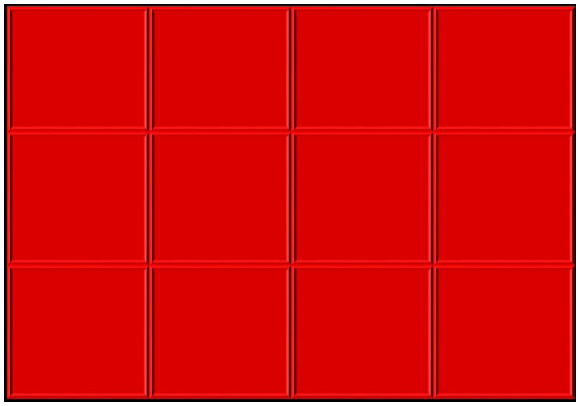
and eventually ....



all tiles are matched,

and the game is over!

1. Like a checkerboard, a Concentration board can be represented as a two-dimensional array. Write the declaration statement to declare a 2D array that represents the following Concentration board:



Check your answer

```
String[ ][ ] gameboard = new String[3][4];
```

Part II: The Game Objects

For your version of the Concentration game, you will place `Tile` objects into a 2D array called `gameboard`. Think of a `Tile` as a picture card. The picture card has a card face and a card back. The card can have a state - it can be either face up or face down - on the playing board.

2. Open the BlueJ project, Concentration.
3. Review the `Tile` class and its instance variables.
4. Now look at its methods. The `Tile` class has objects for `cardFace` (a `String`) , `cardBack` (a literal), and `cardMatched` (another literal). It also has a `faceUp` state variable. which stores whether it is face up or face down, and a matched state variable to store whether it has been matched to another tile.

In the `Tile` constructor, why didn't the programmers set `faceUp` and matched to false?

5. Review the `ConcentrationRunner` class. In the `main` method, the first block of code contains instructions for the game. Most of the work is done in the next construct, a `while` loop. Within this loop,

   - The game board is displayed
   - A player selects a tile
   - The board is displayed again with the selected tile face up
   - The player selects another tile
   - The board is displayed again, this time with both tiles face up
   - Tiles are checked for a match

   This algorithm repeats while there are unmatched tiles on the board.

   a. Where is the variable `game` declared?

   b. How is it initialized?

   c. What are `game's` instance variables and methods?

6. Review the Concentration class, focusing on the javadocs. As you read through them, start to form some ideas of how you might implement some of the algorithms that are described for each method.

Part III: Task 1: Populate the Game Board

7. Summarize how `gameboard` is declared and initialized. Be sure to check any super classes.

8. After initialization, what are the values of the elements in `gameboard`?

Debugging Tips: During game play, the BlueJ console is cleared between plays. This may interfere with debugging statements you want to display. You can temporarily turn off this feature. Open `ConcentrationRunner.java`. In the `displayBoard(`, comment out the following line: `System.out.print('\u000C');`

Comment out portions of working code to make playing the game easier and faster. For example, to test your card matching algorithm, don't hide the cards while you try to "find" matches.

9. Observe the Concentration constructor, where you will begin to create the game algorithms.
10. Populate the 2D `gameboard` array with tiles, using a nested for loop algorithm.
    - Use a test value for each element, such as `new Tile("test")` or `new Tile("*")`
    - Do not hard code the lengths of the array.
    - Compile and run `ConcentrationRunner`. Your program will not print anything yet; the `toString` method for Concentration returns an empty string.

11. Use a similar nested loop algorithm to display the contents of the `gameboard`.

    - Notice how these two coding segments have the same result:

      ```
      Tile t = gameboard[i][j];
      t.getFace();
      // is equivalent to
      gameboard[i][j].getFace();
      ```

      You may choose to implement your algorithm using either style.

    - Put a tab character (`\t`) after each element, and a newline character (`\n`) at the end of each row.
    - Recall that `gameboard` consists of `Tiles`. Use the appropriate `Tile` method to display the face value.

      NOTE

      `Tiles` will eventually contain card values, so the variable names in `Tile` may not seem intuitive at this point.

    - Compile and test. You can run the program or select `displayGame()` from the `ConcentrationRunner` context menu. To quit the game, type any character other than a number.

You should see your game board displayed as a 3 x 4 matrix of test strings.

Part III: Task 2: Replace Test Data with Game Cards

12. Review the `Board` class. Back in `Concentration`, replace your test data with the string values defined in Board's `getCards` method.

> **Hint**: Create a `String` array of cards, copy values from the cards array to the `gameboard`, and track the remaining cards with an index counter.

13. Compile and test. Your board should show the game cards as:

```
dog     dog     cat     cat
mouse   mouse   wolf    wolf
monkey  monkey  bird    bird
```

The cards are on the board, but they need to be randomized to make this a game that is fun to play.

Part III: Task 3: Randomize Game Cards

14. Consider this algorithm that selects random elements from an array until all of the elements have been used:

> You have an array of strings with `num` elements.
>
> *Choose* a random number between 0 and `num` and use that randomly selected element.
>
> *Move* the last element of the array to the element you just used; in Java you are not really *moving*—simply overwrite the randomly selected element with the last element of the array (the element at `num`).
>
> Then *decrement* `num`.
>
> Repeat this *choose-move-decrement* process until all elements of the array have been used.
>
> Notice that even though the last elements of the array still have values assigned to them, you will never use them because you are choosing random numbers between `0` and `num`. This `num` controls the *end* of the array, and you decrement `num` as you use elements.

15. To initialize the `gameboard` in Concentration using this algorithm, randomly populate the board with the tiles of cards from your cards array.
16. Compile and test that the cards are randomized each time you start a new game.

Part III: Task 4: Selecting Cards

17. The next step is to place the cards face down on the table. The `Tile` class has a method for getting the state of a `Tile`. Modify `Concentration.toString()` so it will display each element of `gameboard` face up or face down depending on the value of its state variable (`faceUp`). At this point—the start of the game—all tiles should be face down.

Debugging Tip: You may want to recall the `toString` method. It is an easy change to always show card faces, which may be helpful in testing algorithms and finding bugs.

Now you will code the primary functionality.

18. In `ConcentrationRunner`, after the first call to `displayBoard`, the algorithm collects two integers, `i1` and `j1`. These are the row and column values of the *first* `Tile` in `gameboard`. What are two integer variables for the *second* `Tile`?

19. Implement the `showFaceUp` method in `Concentration` to indicate that a tile at the location specified should be shown face up.

20. Test the game. (Notice there is a pause after you enter your second choice. Eventually, this is to let the player observe the `gameboard` before their turn is over.) You should be able to see all of the tiles on the game board by entering correct values for `i` and `j`.

21. Next, you need to check whether the selected tiles match and take the appropriate action to either turn the cards face down or remove them from the game. Override the `equals` method for `Tile` and implement `Concentration.checkForMatch`.

22. Your game algorithm is almost complete; you just need a check for the end of the game: Implement `allTilesMatch` to determine when the game is over.

Part IV: Enhancing the Game

NOTE

Remember to use the debugging tips mentioned throughout this activity.

23. First, test that your implementation is sound. In `Board.java`, add or remove cards from the cards array and change rows and columns accordingly. Verify that you can play the game with the larger or smaller board.

24. You may have noticed some problems with the game:

    - Since input is not validated, the preconditions for `checkForMatch` and `showFaceUp` are not being met. This can cause the program to crash.
    - Due to another missing input validation, choosing the same card in the same turn makes the game unwinnable.
    - If a card is already matched, it can still be used again.

    Fix these problems as your teacher indicates, or as time allows. You may also choose to correct the problems after the next section, “Sevens”.

Part V: Sevens

25. You will now make a variation of the Concentration game called "Sevens". Instead of comparing strings, this version of the game will add two cards together to see whether they add to 7. Because Concentration is an object-oriented program, the objects can be reused. To create the Sevens version, you will need to make only small changes to your program.

26. In `Board.java`, comment out the old cards array and add this new card array:

```
private String[] cards = new String[] { "1", "1", "2" ,"2",
    "3", "3", "4", "4", "5", "5" ," 6", "6"};
```

27. In `Concentration.java`,

    a. Create two new integer constants, `CONCENTRATION` and `SEVENS`, and assign them different values. (The specific values are not important; just be sure they have different values, such as 100 and 200.)

    b. Create an integer instance variable for `gamerules`.

    c. In the constructor, set gamerules to `SEVENS`.

28. Now, change the way tiles are compared. With `CONCENTRATION` rules, you overwrote the `Tile's` equals method to compare card faces. But for `SEVENS's` rules, you must add the value of the card faces:

    a. In `Tile.java`, create a new method to add card faces together. The algorithm for this method is provided:

```
int a = Integer.parseInt(this.cardFace);
int b = Integer.parseInt(tile.cardFace);
if ( a + b == 7) return true;
else return false;
```

    b. In `checkForMatch`, add code so that if the game is `SEVENS`, the proper match is made.

    c. With these few changes, you should be able to play the game of Sevens. Run and test your code.

29. For a mashup of Concentration and Sevens, change the game rules to `CONCENTRATION` and make no other changes.

30 Run and play the game. How are the rules mashed? Can you think of an improvement to the game related to this mashup?

AP Focus: **FRQ: GrayScale**

CONCLUSION

1. If the `Tile` object does not have an `equals` method, explain the logic of the following algorithm and validate or invalidate it:

```
Tile t1 = new Tile("hippo");
Tile t2 = new Tile("hippo");
if (t1.equals(t2)) {
    // tiles match!
}
```