IQ Focus Game

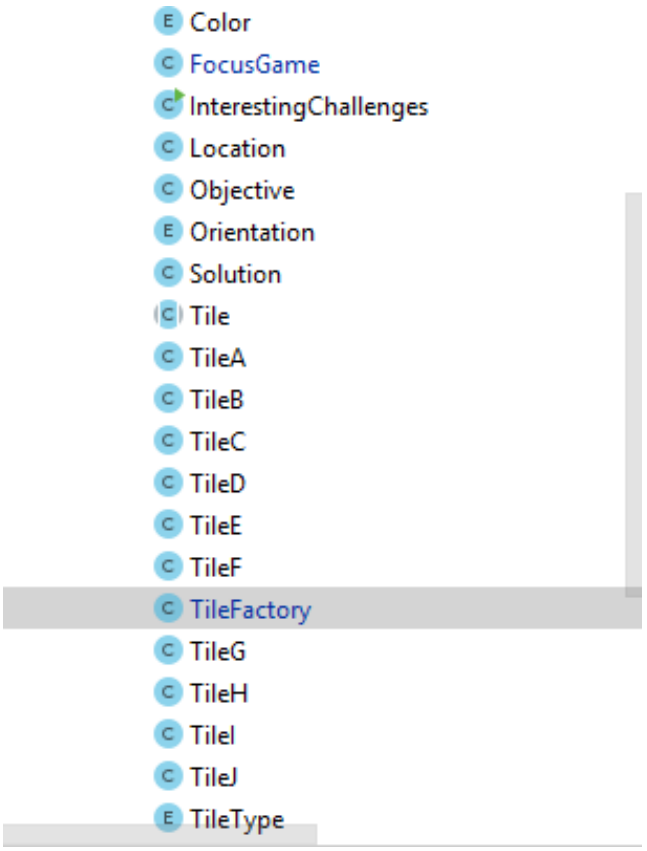
By

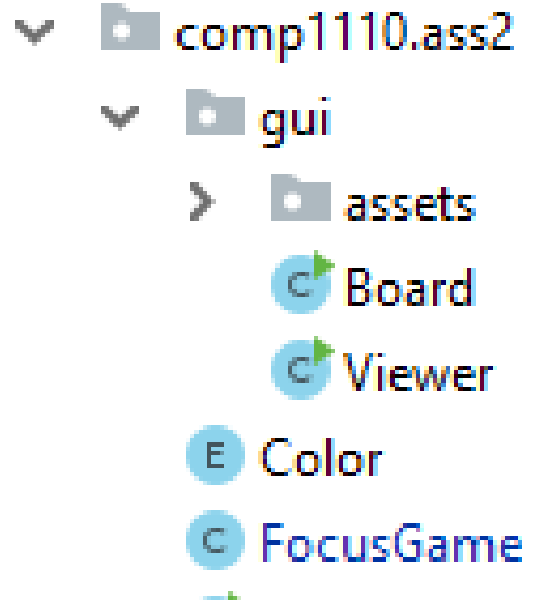
Tanya Dixit

Samuel H Brown

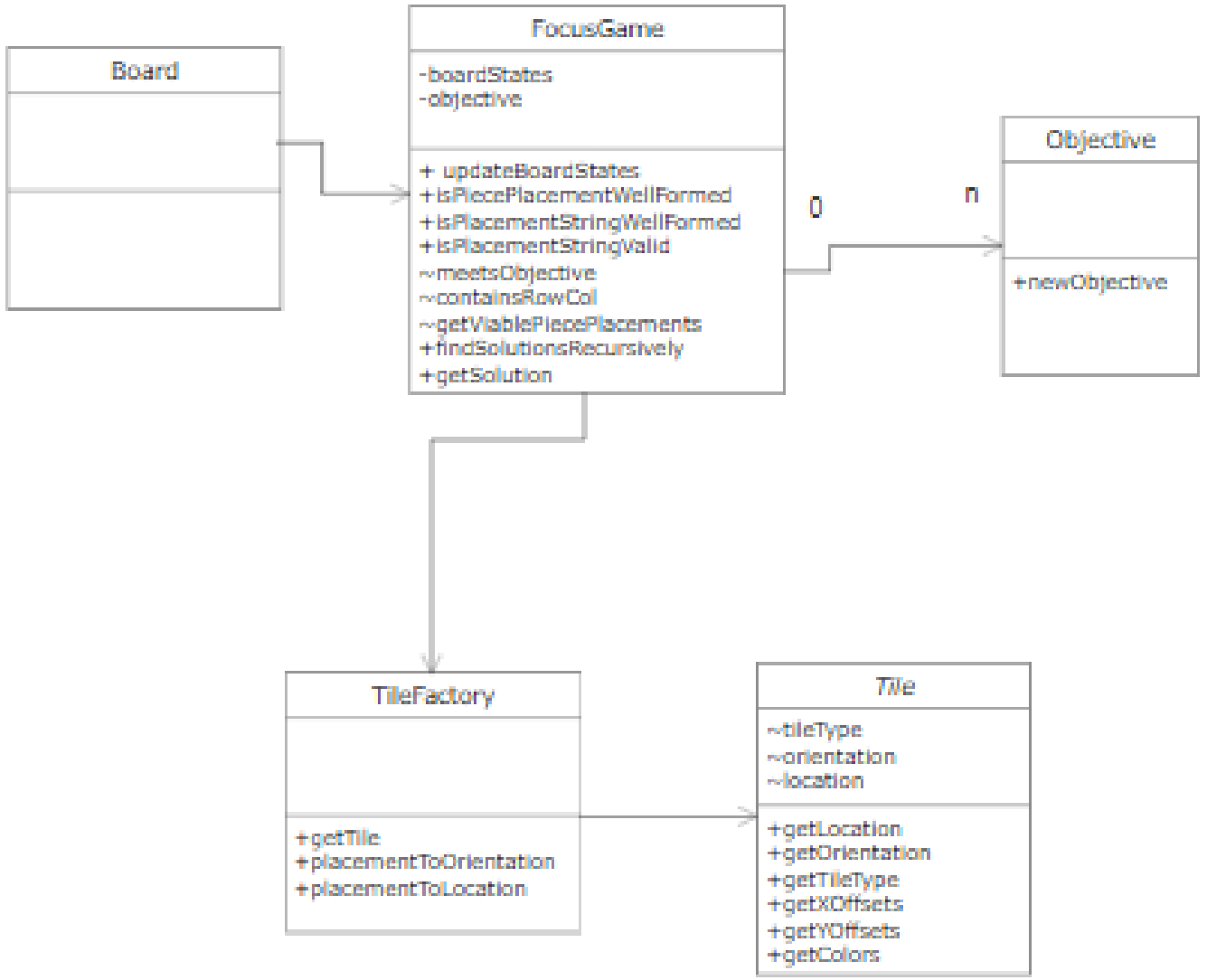
Mahesh Gundubogula

Class Design





Class Diagram



Features

A basic playable game

Simple challenges according to difficulty

Interesting generated challenges

Backend Design and Features

Design considerations – Fast and scalable

- Factory Design Pattern
  - Made the code smaller and easier to read/write
  - Abstracted away tile information from the main class
- boardStates – a 2-D Array of Color enum to represent game state
  - EMPTY and FORBIDDEN to denote empty place and not allowed
  - R,G,B,W to denote colors

```java
 */

public abstract class Tile {

    /*
     * A tile has a tileType (see TileType enum)
     *            an orientation (see Orientation enum)
     *            a location (see Location class)
     */

    TileType tileType;
    Orientation orientation;
    Location location;

    public Location getLocation() { return location; }

    public Orientation getOrientation() { return orientation; }

    public TileType getTileType() { return tileType; }

    public abstract int[] getXOffsets();
    public abstract int[] getYOffsets();
    public abstract Color[] getColors();

}
```

```java
public class TileA extends Tile {

    /*
     * Color array tells what colors are in the tile
     * xOffsets are according to color and orientation.
     *
     * So for example, if we want to find where W is in TileA,
     * for orientation 1 it will be x+1
     *
     * yOffsets are just previous orientation's xOffsets
     */
    private Color[] colorArray = {G,W,R,R};
    private int[][] xOffsets = {{0,1,2,1},{1,1,1,0},{2,1,0,1},{0,0,0,1}};

    public TileA (TileType tileType, Orientation orientation, Location location) {
        this.tileType = tileType;
        this.orientation = orientation;
        this.location = location;
    }

    @Override
    public int[] getXOffsets() { return xOffsets[this.orientation.getValue()]; }

    @Override
    public Color[] getColors() { return colorArray; }

    @Override
    public int[] getYOffsets() {
        int length = 4; //number of orientations

        /*
         * From observation, we found that yOffsets were just previous
         * orientations xOffsets
         */
```

```java
public static Tile getTile(String placement) {

    TileType tileType = TileType.valueOf(Character.toString((placement.charAt(0) - 32)));
    Orientation orientation = placementToOrientation(placement);
    Location location = placementToLocation(placement);

    //return suitable tile object based on the TileType
    switch (tileType) {
        case A:
            Tile newTileA = new TileA(tileType, orientation, location);
            return newTileA;
        case B:
            Tile newTileB = new TileB(tileType, orientation, location);
            return newTileB;
        case C:
            Tile newTileC = new TileC(tileType, orientation, location);
            return newTileC;
        case D:
            Tile newTileD = new TileD(tileType, orientation, location);
            return newTileD;
        case E:
            Tile newTileE = new TileE(tileType, orientation, location);
            return newTileE;
        case F:
            Tile newTileF = new TileF(tileType, orientation, location);
            return newTileF;
        case G:
            Tile newTileG = new TileG(tileType, orientation, location);
            return newTileG;
        case H:
            Tile newTileH = new TileH(tileType, orientation, location);
            return newTileH;
```

```java
public class TileFactory {

    /*
     * Return a tile object based on the placement passed to it.
     * This function looks at the placement string, decides which type
     * it is by looking at the first character, and creates and returns
     * a suitable tiletype object.
     *
     * @param placement: A string which has tiletype + x + y + orientation
     * @returns newTile: A Tile object based on the placement
     */
    public static Tile getTile(String placement) {

        TileType tileType = TileType.valueOf(Character.toString((placement.charAt(0) - 32)));
        Orientation orientation = placementToOrientation(placement);
        Location location = placementToLocation(placement);

        //return suitable tile object based on the TileType
        switch (tileType) {
            case A:
                Tile newTileA = new TileA(tileType, orientation, location);
                return newTileA;
            case B:
                Tile newTileB = new TileB(tileType, orientation, location);
                return newTileB;
```

Challenges

getSolution taking a lot of time

Reason:

State boardStates is not preserved

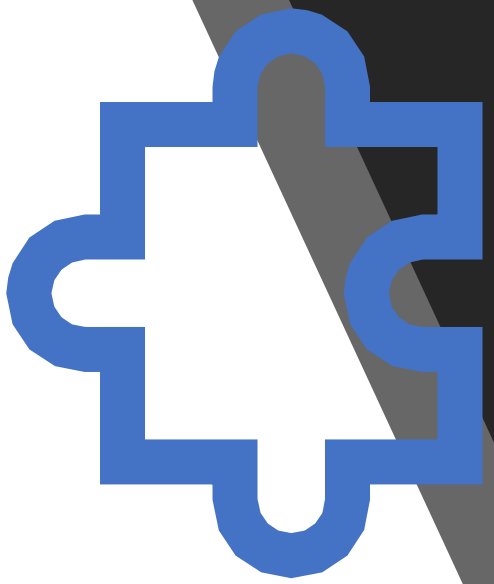
Every time a new game created when we place a tile

Couldn’t find a way around it

```java
public static String findSolutionsRecursively(String placedNow, String challenge) {

    int[] rowCol = {-1,-1};
    Color[][] boardStates = {
            {EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY},
            {EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY},
            {EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY},
            {EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY},
            {FORBIDDEN, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, FORBIDDEN}
    };

    for (int i = 0; i < placedNow.length()/4; i++) {
        Tile tile;
        tile = TileFactory.getTile(placedNow.substring(4 * i, 4 * i + 4));

        int[] xOffsets = tile.getXOffsets();
        int[] yOffsets = tile.getYOffsets();
        Color[] colors = tile.getColors();
        int x = tile.location.getX();
        int y = tile.location.getY();

        for (int j = 0; j < xOffsets.length; j++) {
            boardStates[y + yOffsets[j]][x + xOffsets[j]] = colors[j];
        }
    }

    //This gives row and col number that's not occupied.
    //We first search in the objective array
    for (int i = 0; i < objectiveX.length; i++) {
        if (boardStates[objectiveY[i]][objectiveX[i]] == EMPTY) {
            rowCol[0] = objectiveX[i];
            rowCol[1] = objectiveY[i];
```

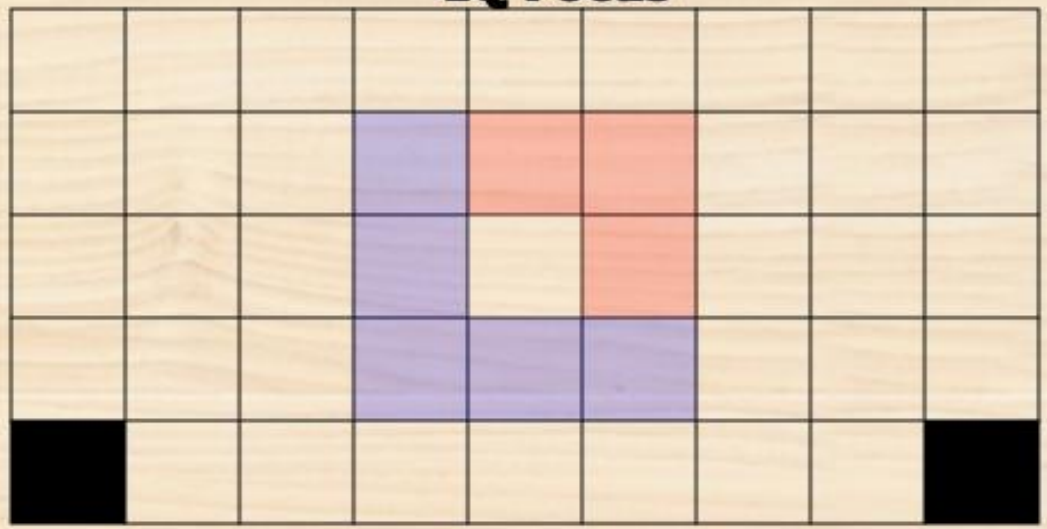
```java
    /*
     * If all positions are occupied, we got our solution
     */
    if (rowCol[0] == -1 && rowCol[1] == -1)
        return placedNow;

    Set<String> currentViablePlacements = getViablePiecePlacements(placedNow, challenge, rowCol[0], rowCol[1]);

    if (currentViablePlacements == null) {
        return null;
    }

    /*
     * Loop over the viable placememts and check if more viablePlacements are there
     * eventually leading to solution
     */
    for (String s: currentViablePlacements) {

        String x = findSolutionsRecursively( placedNow: placedNow + s, challenge);
        if (x == null) {
            //If no solutions with this string, don't add it, return null
            continue;
        } else {
            //return current string appended by x
            return s+x;
        }
    }

    return null;
}
```

Frontend Design and Features



- Objective as part of the Grid
- Reset button
- Restart button – restarts the game with a new objective
- Difficulty slider
- Challenge me more!!! – Creates a new game with objective from the saved interesting challenges
- Tooltips

IQ Focus

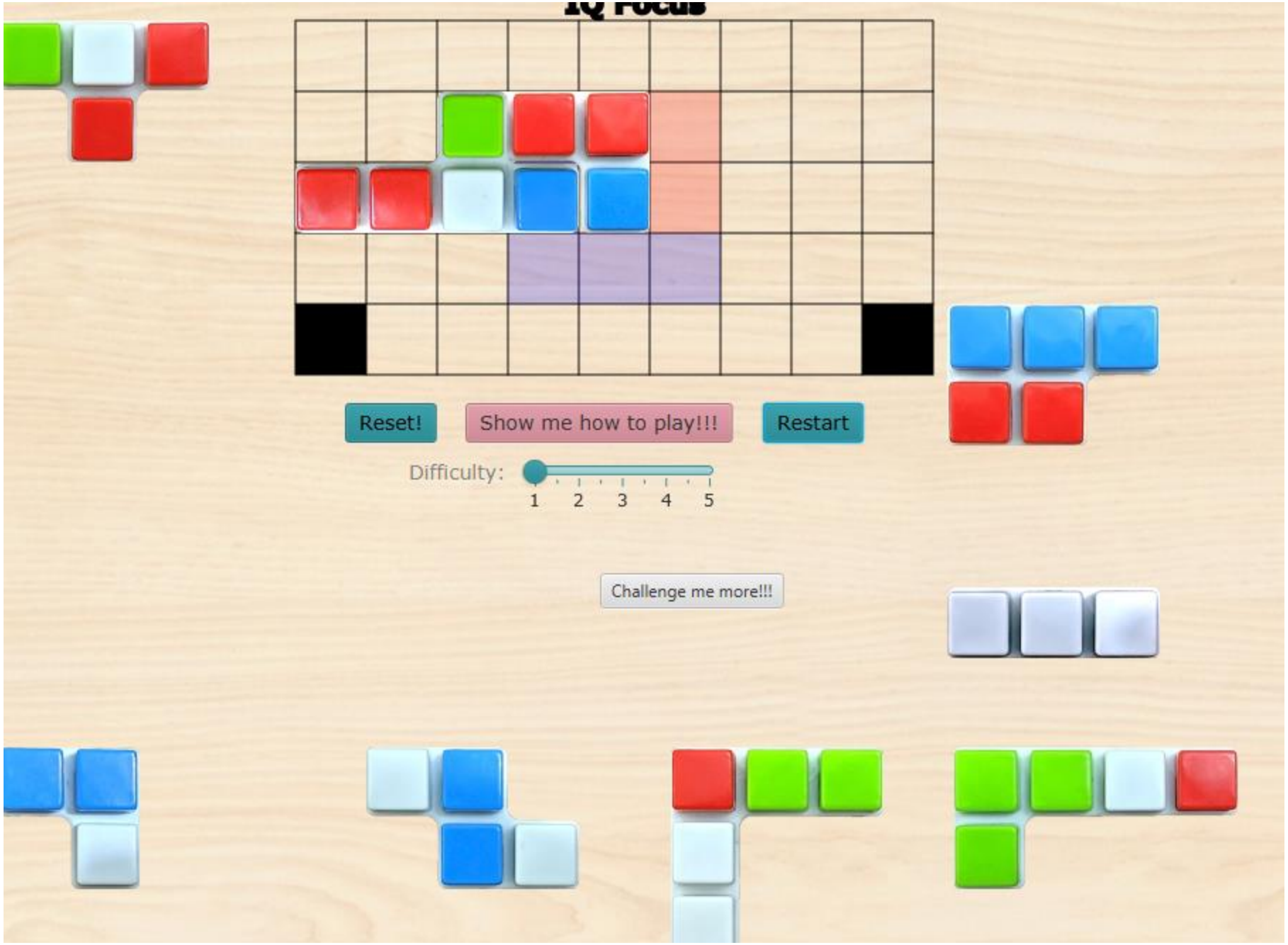


Reset! Show me how to play!!! Restart

Difficulty: 1 2 3 4 5

Challenge me more!!!

Challenges


IQ Focus
Reset!
Show me how to play!!!
Restart
Difficulty:
1 2 3 4 5
Challenge me more!!!

```java
/*
 * Challenge Type 2: Three consecutive tiles are same color. For e.g.
 * RRRWWWBBB or WWWGGGRRR.
 * This function has a side effect of adding or appending the challenges and their solutions
 * in the interestingSol HashMap.
 *
 * @param: void
 * @returns: void
 */
public void addChallengeType2() {
```

```java
static Map<String, String> interestingSol = new HashMap<>();
```

```java
/*
 * Challenge Type 1: All the nine tiles are same color. For e.g.
 * RRRRRRRRR or WWWWWWWWW.
 * This function has a side effect of adding the challenges and their solutions
 * in the interestingSol HashMap.
 *
 * @param: void
 * @returns: void
 */
public void addChallengeType1() {
```

```java
/*
 * Challenge Type 3: Three consecutive tiles are same color vertically. For e.g.
 * RWB    GWR
 * RWB or GWR
 * RWB    GWR
 *
 * This function has a side effect of adding or appending the challenges and their solutions
 * in the interestingSol HashMap.
 *
 * @param: void
 * @returns: void
 */
public void addChallengeType3() {
```

```java
/*
 * Challenge Type 4: Diagonals have same colors. For e.g.
 * RWB
 * WRW
 * BWR
 *
 * This function has a side effect of adding or appending the challenges and their solutions
 * in the interestingSol HashMap.
 *
 * @param: void
 * @returns: void
 */
public void addChallengeType4() {
```

Interesting challenges

Best code features

- Code is scalable
- Easy to understand
- Less repetition
- Works fast
- Game looks good
- Easy for a user to understand

The Not-so good parts

- Recursive algorithm for solutions is slow
- Front-end not connected to Back-end -> Board.java not connected to FocusGame.java
- Overlap checks don’t work. Would be easier if the above connection was present

Thank you