IQ-Fit Game

By

Jiwon Sin

Di Mou

Mingxuan Wang
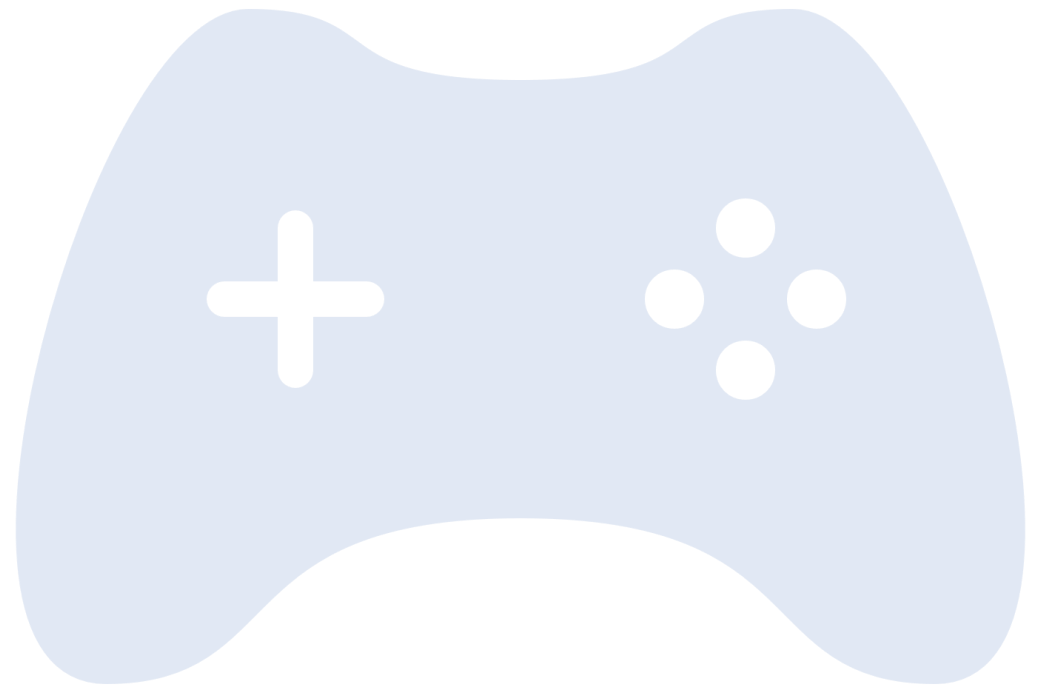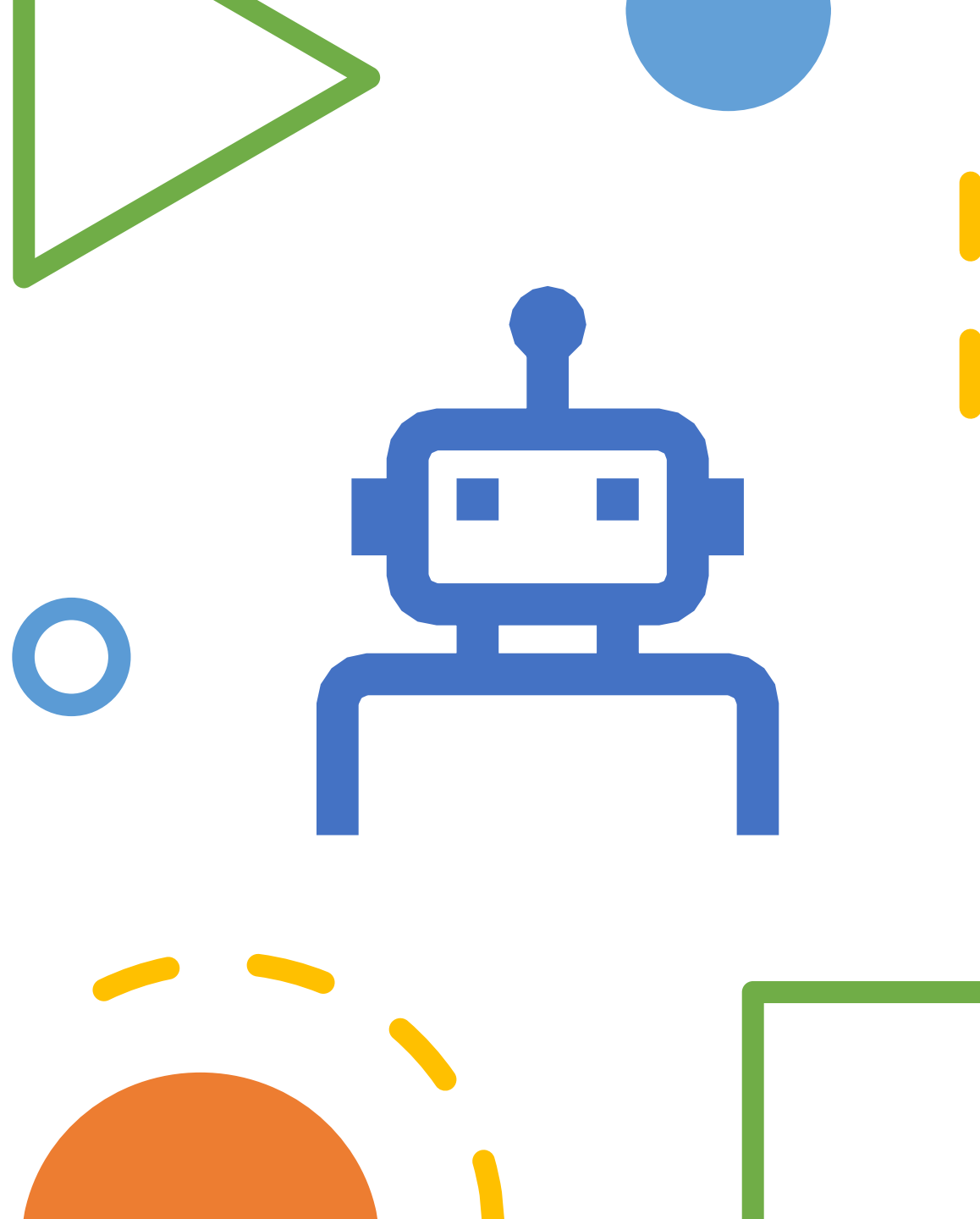
Contents

Class and Method Design

- Problems
  - Piece definition
  - Board design
  - Viewer design (mainly JavaFX)
  - Implementing methods to game
  - How to improve getSolution()

```
blic enum PieceType {

    B(blue,    protrusion: 2, spineNum: 4), b(blue,    protrusion:
    G(green,   protrusion: 2, spineNum: 3), g(green,   protrusio
    I(indigo,  protrusion: 2, spineNum: 3), i(indigo,  protru
    L(limegreen,   protrusion: 2, spineNum: 3), l(limegreen,
    N(navyblue,    protrusion: 2, spineNum: 3), n(navyblue,
    O(orange,  protrusion: 2, spineNum: 4), o(orange,  protru
    P(pink,    protrusion: 2, spineNum: 4), p(pink,    protrusion:
    R(red,     protrusion: 2, spineNum: 4), r(red,     protrusion: 1,
    S(skyblue,     protrusion: 2, spineNum: 4), s(skyblue,   pro
    Y(yellow,  protrusion: 2, spineNum: 4), y(yellow,  protru


    public final PieceColour colour;
    public final int protrusion;
    public final int spineNum;
```

```
ass Piece {

c final PieceType type;
c final PieceDirection dir;
c final PieceCoordinates coords;


e pieces we set has three different elements.
aram type Defines the piece's type, number of protrusions an
aram coords Defines the piece's direction, whether its N,S,E
aram dir Defines where the piece is located on the board.


de written by Jiwon Sin


  (PieceType type, PieceCoordinates coords, PieceDirection di
his.type = type;
his.coords = coords;
his.dir = dir;
```

```
public enum PieceDirection {
    NORTH( symbol '↑'), SOUTH( symbol '↓'), EAST( symbol '→'), WEST( symbol '←');

    public char symbol;


    /**
     * Constructor for piece's direction
     * @param symbol The symbol (in direction) of the piece
     *
     * Code written by Jiwon Sin
     */
    PieceDirection(char symbol) { this.symbol = symbol; }
```

```
public final int xCoord;
public final int yCoord;

/**
 * Constructor for piece's coordinate
 *
 * @param x Value of X coordinate
 * @param y Value of Y coordinate
 *
 * Code written by Jiwon Sin
 */

PieceCoordinates(int x, int y) {
    this.xCoord = x;
```
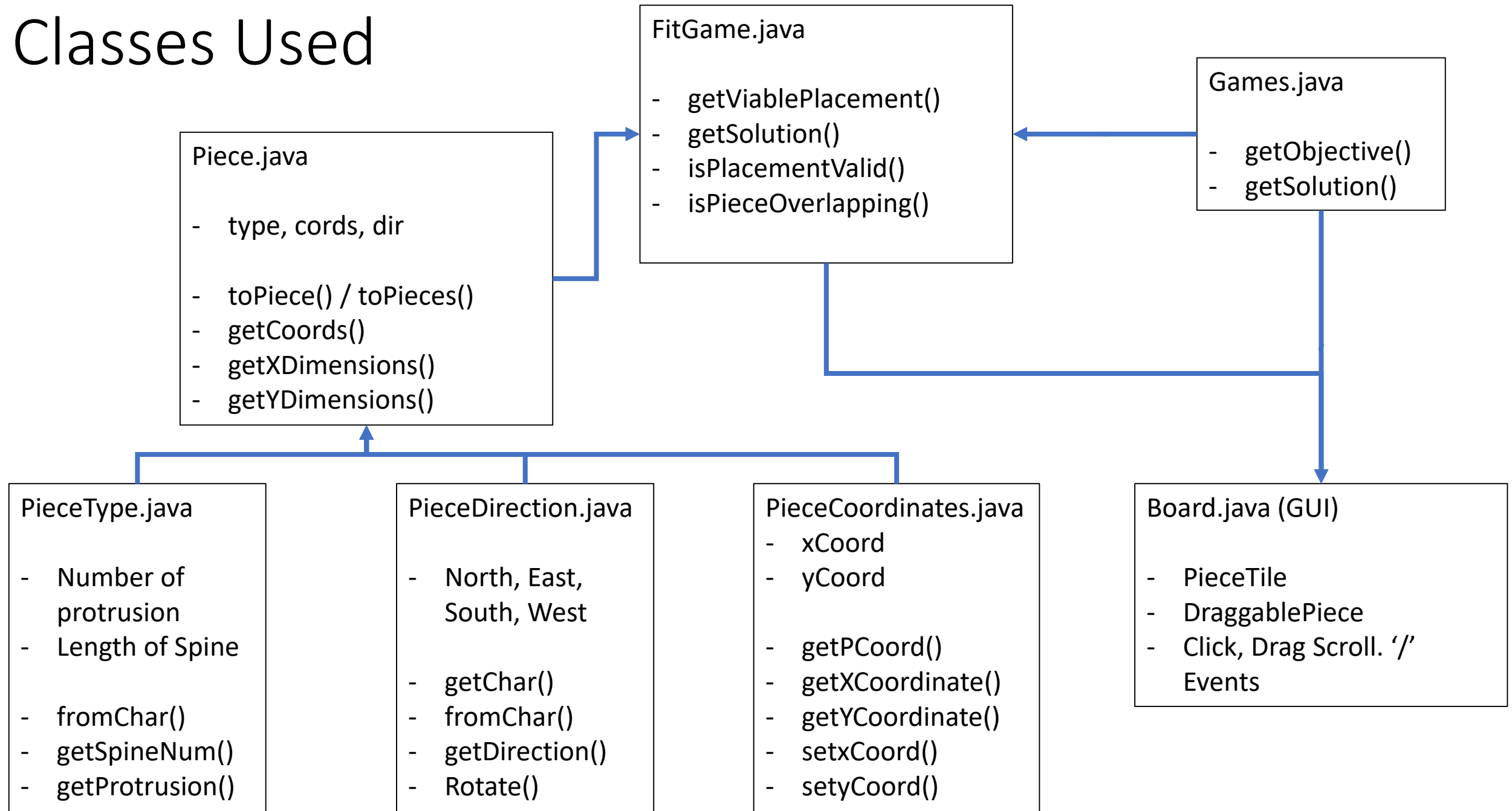
```
ublic enum PieceColour
    blue, green,
    indigo, limegreen,
    navyblue, orange,
    pink, red,
    skyblue, yellow
```

Piece and Board definition

- Piece comprises of
  - Type
    - Colour
    - Protrusion
    - spineNum
  - Coordinates
  - Direction

- Converting String to Piece

- How to apply these Pieces to gameboard?

Classes Used

FitGame.java

- getViablePlacement()
- getSolution()
- isPlacementValid()
- isPieceOverlapping()

Games.java

- getObjective()
- getSolution()

Piece.java

- type, cords, dir

- toPiece() / toPieces()
- getCoords()
- getXDimensions()
- getYDimensions()

PieceType.java

- Number of protrusion
- Length of Spine

- fromChar()
- getSpineNum()
- getProtrusion()

PieceDirection.java

- North, East, South, West

- getChar()
- fromChar()
- getDirection()
- Rotate()

PieceCoordinates.java
- xCoord
- yCoord

- getPCoord()
- getXCoordinate()
- getYCoordinate()
- setxCoord()
- setyCoord()

Board.java (GUI)

- PieceTile
- DraggablePiece
- Click, Drag Scroll. '/' Events

Key Components : Board and Pieces

How to convert placement to Piece?

Use getCoords()

Checks the colour and direction

How to define game board?

Use 2-Dimensional Array

PieceType [][] initialBoard

How to apply Piece to game board?

```java
Piece[] pieces = toPieces(placement);

PieceType[][] initialBoard = {
        {null, null, null, null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null, null, null, null}
};

for (Piece piece : pieces) {
    PieceType[][] array = piece.getCoords();
    int x = piece.coords.getXCoordinate();
    int y = piece.coords.getYCoordinate();
    for (int j = x; j < x + piece.getXDimensions(); j++) {
        for (int i = y; i < y + piece.getYDimensions(); i++) {
            if (initialBoard[i][j] != null && array[i - y][j - x] != null) {
                return false;
            }
            if (array[i - y][j - x] != null) {
                initialBoard[i][j] = array[i - y][j - x];
            }
        }
    }
}
return true;
```

Key Components : GUI

PieceTile

Checks whether its valid type

DraggablePiece

Obtain pieces via setPlayablePieces()

makePieces() constructs pieces graphically

Different events update pieceID

snapToGrid()

Calculates x and y values based on x and y coordinates on the board

Snapping animation

Range of values

PieceTiles

```java
class DraggablePiece extends PieceTile {
    double homeX, homeY;
    double mouseX, mouseY;
    int orientation; // 0 = NORTH, 1 = EAST 2 = SOUTH 3 = WEST
    int positionX, positionY;
    char type;

    DraggablePiece(String placement) {
        super(placement);
        type = placement.charAt(0);
        orientation = 1;
        char piece = placement.charAt(0);

        Image pieceImage;

        positionX = Character.getNumericValue(placement.charAt(1));
        positionY = Character.getNumericValue(placement.charAt(2));

        if (Character.isLowerCase(type))
            pieceImage = new Image(getClass().getResource( name: URI_BASE + Character.toUpperCase(placement.charAt(0)) +"1.png").toString());
        else
            pieceImage = new Image(getClass().getResource( name: URI_BASE + (placement.charAt(0)) +"2.png").toString());
        setImage(pieceImage);
```

```java
static class PieceTile extends ImageView {
    String pieceID;

    PieceTile(String piece) {
        char [] pieceArray = {'b','g','i','l','n','o','p','r','s','y'};
        char [] pieceArrayUp = {'B','G','I','L','N','O','P','R','S','Y'};
        for (int i = 0; i < pieceArray.length; i++){
            if ((piece.charAt(0) == pieceArray[i] || piece.charAt(0) == pieceArrayUp[i])) {
                break;
            }
            else {
                if (i == 9) {
                    throw new IllegalArgumentException("Bad piece: " + piece + " at " + i);
                }
            }
        }
        this.pieceID = piece;
        setFitWidth(SQUARE_SIZE * fromChar(piece).getSpineNum());
        setPreserveRatio(true);
    }
}
```

```java
// Handling events

    setOnMousePressed(mouseEvent -> {
        mouseX = mouseEvent.getSceneX();
        mouseY = mouseEvent.getSceneY();
        toFront();
        if (FitGame.isPlacementValid(pieceID)) {
            clearInitialBoard(pieceID);
            Board.addedPieces.remove(pieceID);
        }
        mouseEvent.consume();
    });

    setOnMouseDragged(mouseEvent -> {
        double movementX = mouseEvent.getSceneX() - mouseX;
        double movementY = mouseEvent.getSceneY() - mouseY;
        setLayoutX(getLayoutX() + movementX);
        setLayoutY(getLayoutY() + movementY);
        mouseX = mouseEvent.getSceneX();
        mouseY = mouseEvent.getSceneY();
        mouseEvent.consume();
    });

    setOnScroll(scrollEvent -> {
        if (!isPieceOnBoard()) {
            setRotate((orientation) * 90);
            orientation++;
            setFitWidth(getPieceSpineNum(pieceID) * SQUARE_SIZE);
            setPreserveRatio(true);
            if (orientation == 5) {
                orientation = 1;
            }
        }
    });
```
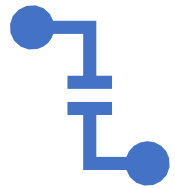
```java
    setOnMouseReleased(mouseEvent -> {
        updatePieceID();
        if (FitGame.isPlacementValid(pieceID)) {
            if (FitGame.isPlacementNotOverlapping(initialBoard, pieceID)) {
                snapToGrid();
                FitGame.boardUpdate(pieceID, initialBoard);
                Board.addedPieces.add(pieceID);
                if (isItComplete()) {
                    showCompletionText();
                    makeCompletionText();
                }
            }
            else {
                setLayoutX(homeX);
                setLayoutY(homeY);
            }
        }
        else {
            setLayoutX(homeX);
            setLayoutY(homeY);
        }
        mouseEvent.consume();
    });
```

Events

Problem and Challenges

- Connecting backend programming and GUI
  - Methods in FitGame.java
  - GUI in Board.java

- Increasing efficiency
  - getViablePiecePlacements()
  - getSolution()
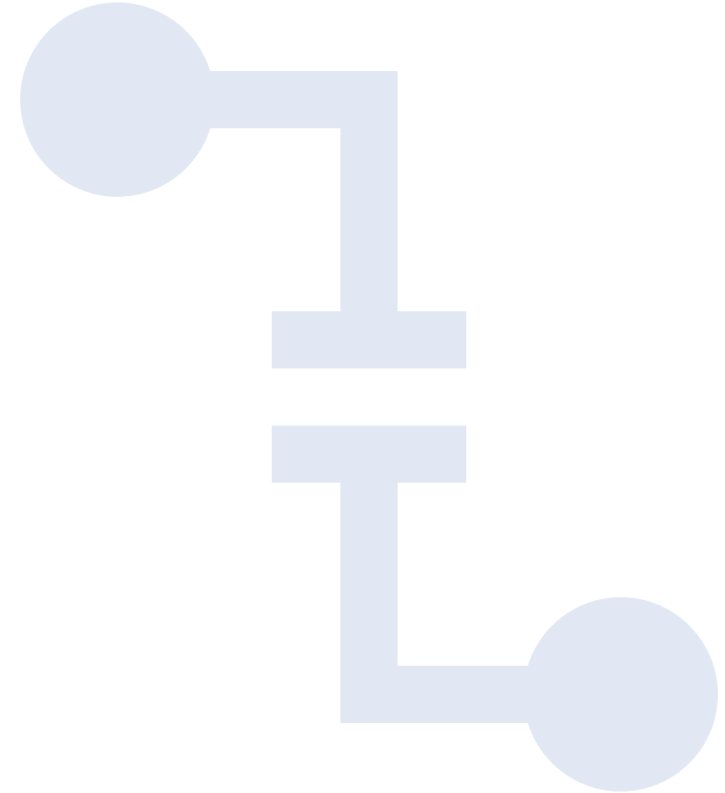
GUI and backend programming

Check whether the piece placement is valid

Not overlapping board

- isPlacementNotOverlapping()

Updating board

- boardUpdate()

Check validity of certain placement

- isPlacementValid()

Use PieceType [][] initialBoard

Increase efficiency

```
public static boolean isThisLogical(String challenge, PieceType [][] initialBoard) {
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 10; j++) {
            if (initialBoard[i][j] == null) {
                if (getViablePiecePlacements(challenge,j,i) == null)
                    return false;
            }
        }
    }
    return true;
}
```

getViablePiecePlacements()

- Diversify conditions
- Multiple methods

getSolution()

- Check whether its "logical"
- Is it possible to have *null* surrounded by PieceTypes?
  - isThisLogical()
- Check which one has least possible choices
  - findOptimalX()

```
public static List<Integer> findOptimalX(String challenge, PieceType [][] board) {
    int possiblePieces = 100;
    int chances;
    List<Integer> array = new ArrayList<>();
    Set<String> numPieces;
    for (int i =0; i < 5; i++) {
        for (int j = 0; j < 10; j++) {
            if (board[i][j] == null) {
                numPieces = getViablePiecePlacements(challenge, j, i);
                if (numPieces != null) {
                    chances = numPieces.size();
                    if (numPieces.size() < possiblePieces) {
                        possiblePieces = chances;
                        array.clear();
                        array.add(i);
                        array.add(j);
                    }
                }
            }
        }
    }
    return array;
}
```
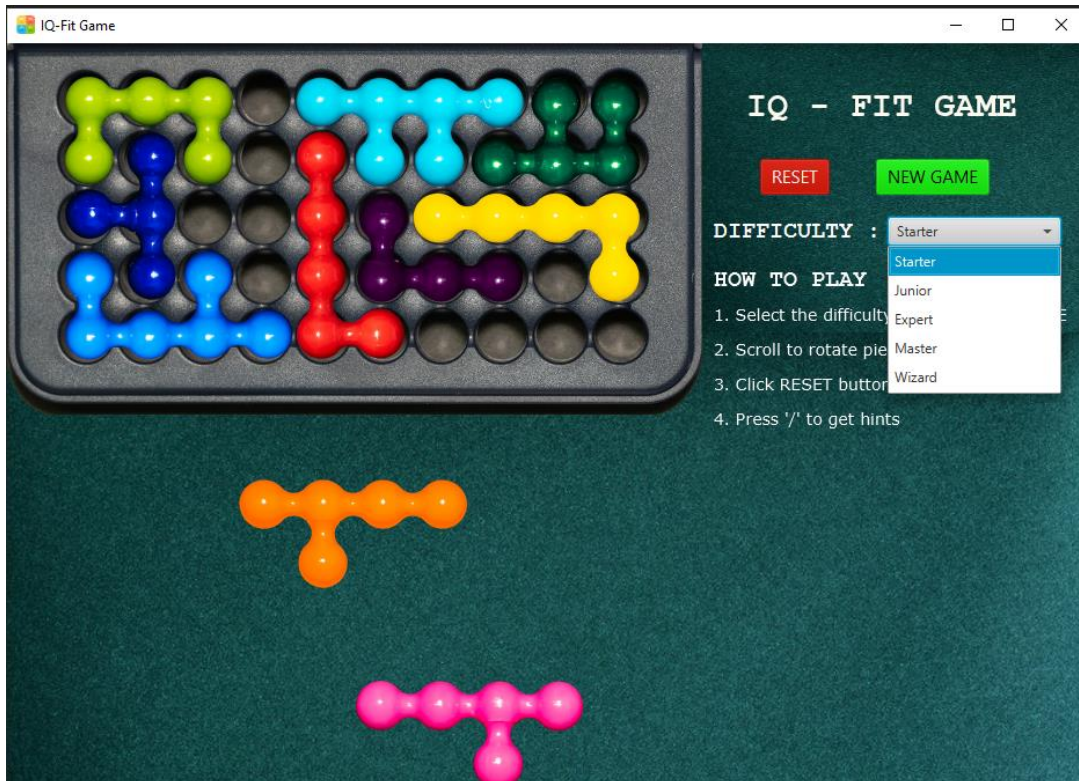
Game
Demonstration
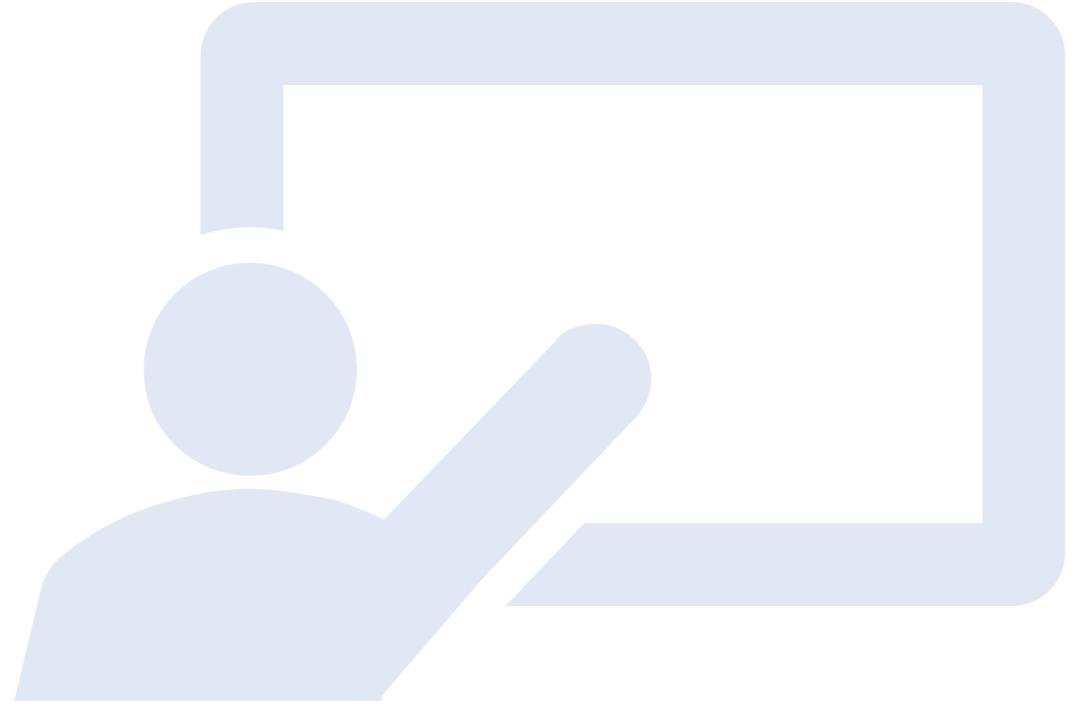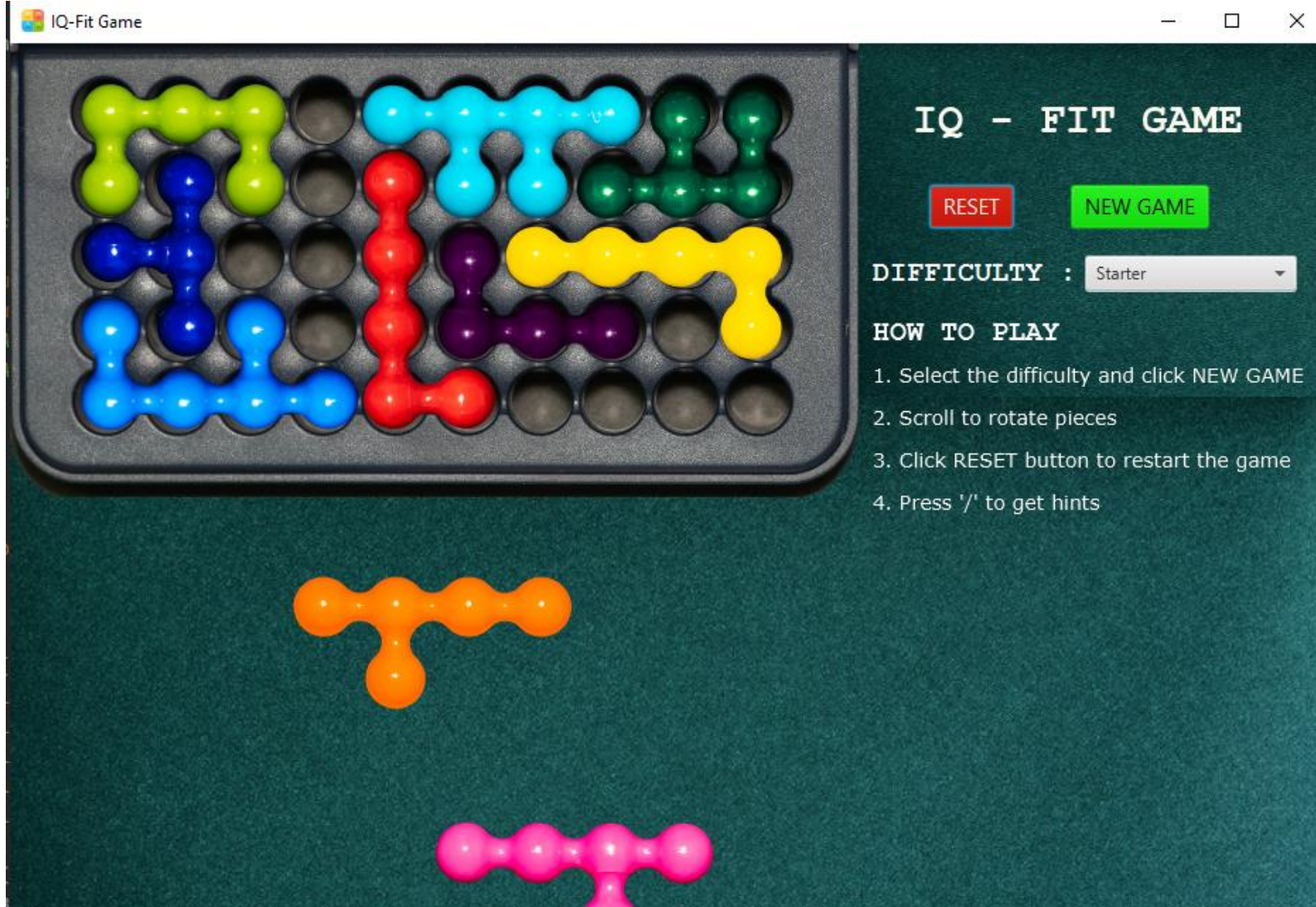
Start of the IQ-Fit Game

Select Difficulty

- 5 Different Difficulty Levels

- Starter
- Junior
- Expert
- Master
- Wizard
- The game starts with "Starter" difficulty as a default

Buttons

- NEW Game
  - Choose a new game but with same difficulty level
  - Game selected in random (0 – 24 random number)

- RESET
  - Resets the pieces
  - When users want to reset the stage

- Getting hints
  - Press '/' button each time

End of Game

- At the end of game
  - Pieces placed does not move when the game ends
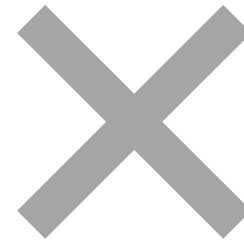 - Player can choose to play another game.
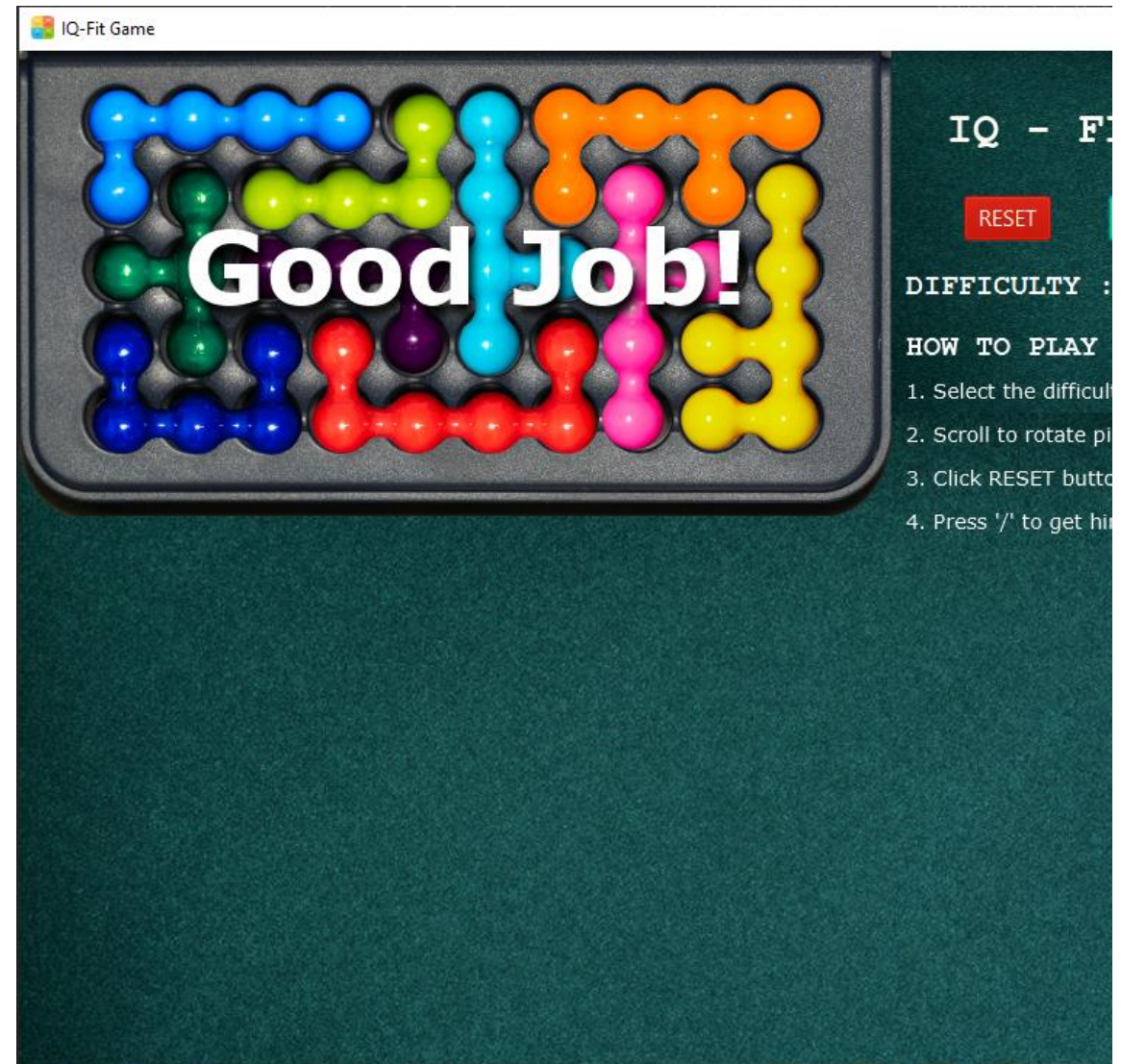
Reflections

Regrettably, getSolution() takes too much time

If difficulty level is at Wizard and Master, test times out.

getValidPiecePlacements() tests duplicates

Timeout error