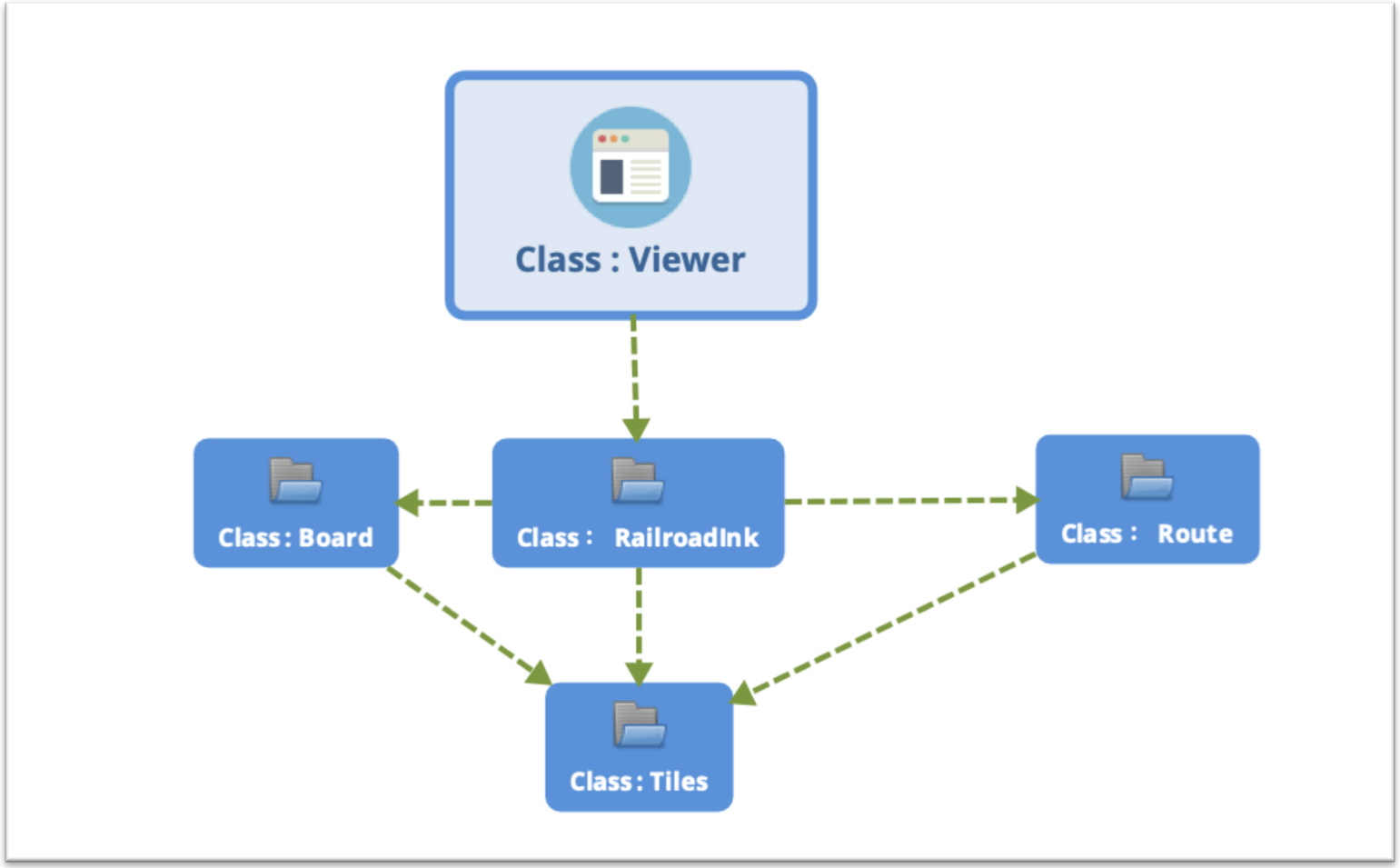
Assignment 2
Railroad Ink: Deep Blue Edition

Jihirshu Narayan (u6811576)
Tuo Liu (u6567008)
Jingfen Qiao (u6843015)

Table of content

Game Structure



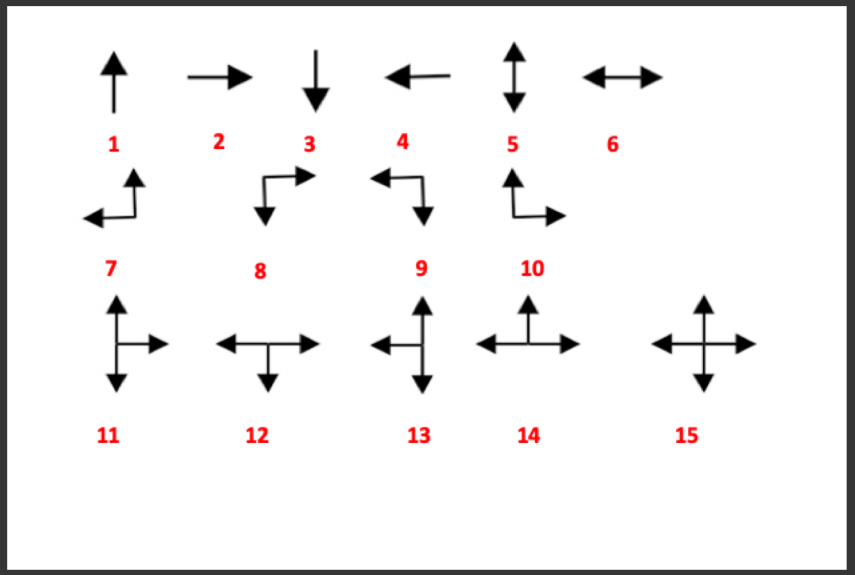
Class Tile

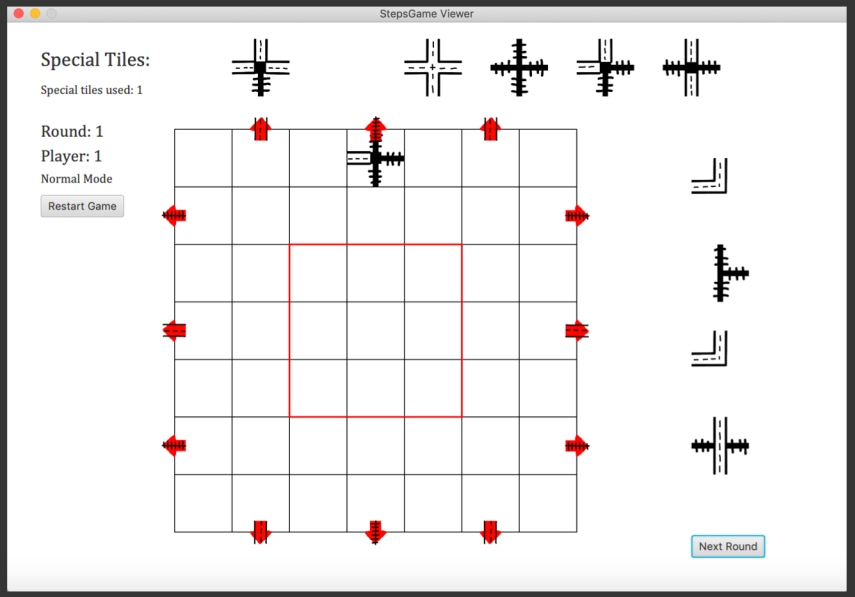
Instance Fields : String name, int[] shape

Encoding Tiles and location

- **Example : S1A37 = {2,0,3,7,4,11}**

- **2**: Tile identity
- **0**: Location on the board
- **3**: Location on the board
- **7**: Orientation of Tile
- **4**: Orientation of High-way
- **11**: Orientation of Railway





Tile Class:
Important functions

- rotate90(int n)
- translate(int x, int y)
- areNeighbours()
- isConnected(Tile x)
- checkRailwayConnection(Tile x)
- CheckHighwayConnection(Tile x)
- checkExitConnection()
- checkIncompatibleConnection(Tile x)
- checkInvalidExitConnection()

```
public boolean isConnected(Tile x)
{
    if (x == null)
        return false;
    boolean result = false, result_railway=false, result_highway=false;
    if ((this.shape[4] * x.shape[4] == 0) && (this.shape[5] * x.shape[5] == 0))
        result = false;
    else if (!areNeighbours(x))
    {
        result = false;
    }
    else
    {
        if ((this.shape[4] * x.shape[4] != 0))
        //highway
        {
            result_highway = this.checkHighwayConnection(x);
        }
        if ((this.shape[5] * x.shape[5]) != 0)
        //railway
        {
            result_railway = this.checkRailwayConnection(x);
        }
    }
    result = result_highway || result_railway;
    return result;
}
```

isConnected()

checkHighwayConnection ()

```
public boolean checkHighwayConnection(Tile x)
{
    Integer[] left = {4,6,10,7,12,13,14,15};
    Integer[] right = {2,6,8,9,11,12,14,15};
    Integer[] down = {3,5,9,10,11,12,13,15};
    Integer[] up = {1,5,7,8,11,13,14,15};
    Tile w = this;
    if (w.shape[1] == x.shape[1])
    {
        if ((w.shape[2] - x.shape[2]) == 1) //if w is on immediate right of x
        {

            if (Arrays.asList(right).contains(x.shape[4]))
            {

                if (Arrays.asList(left).contains(w.shape[4]))
                    return true;
            }
        }
        else if (x.shape[2] - w.shape[2] == 1) //if w is on the immediate left of x
        {
            if (Arrays.asList(left).contains(x.shape[4]))
            {
                if (Arrays.asList(right).contains(w.shape[4]))
                    return true;
            }
        }
    }
    else
    {
        if ((w.shape[1] - x.shape[1]) == 1) //if w is immediately below x
        {

            if (Arrays.asList(down).contains(x.shape[4]))
            {

                if (Arrays.asList(up).contains(w.shape[4]))
                    return true;
            }
        }
        else if (x.shape[1] - w.shape[1] == 1) //if w is immediately above x
        {
            if (Arrays.asList(up).contains(x.shape[4]))
            {
                if (Arrays.asList(down).contains(w.shape[4]))
                    return true;
            }
        }
    }
    return false;
}
```

Class Route

Instance Fields

Tile[] route

- An array of tiles which belonging to a route
- Important functions :
  - addRoute(Tile x)
  - connectedToRoute(Tile x)
  - numberOfExitsConnected()
  - checkRoutesConnected(Route x)
  - mergeRoutes(Route x)

Class Board

Instance Fields

Object[][] board

ArrayList<String> SpecialTiles

int[][] highwayConnections

int[][] railwayConnections

int longestHighway

Int longestRailway

- Variable board contains instances of class Tile
- highwayConnections and railwayConnections is a map representing the highway and railway routes
- Important Functions:
  - checkInvalidConnection(Tile x)
  - centreTileScore()
  - countErrors()
  - countSpecialTiles()
  - generateMoves(String[] choices)
  - checkHighwayConnection(Tile x) and checkRailwayConnection(Tile x)
  - findRailwayNeighboursRecursively(Tile x, int count, int[][] map) and findHighwayNeighboursRecursively(Tile x, int count, int[][] map)
  - getLongestHighwayCount() and getLongestRailwayCount()

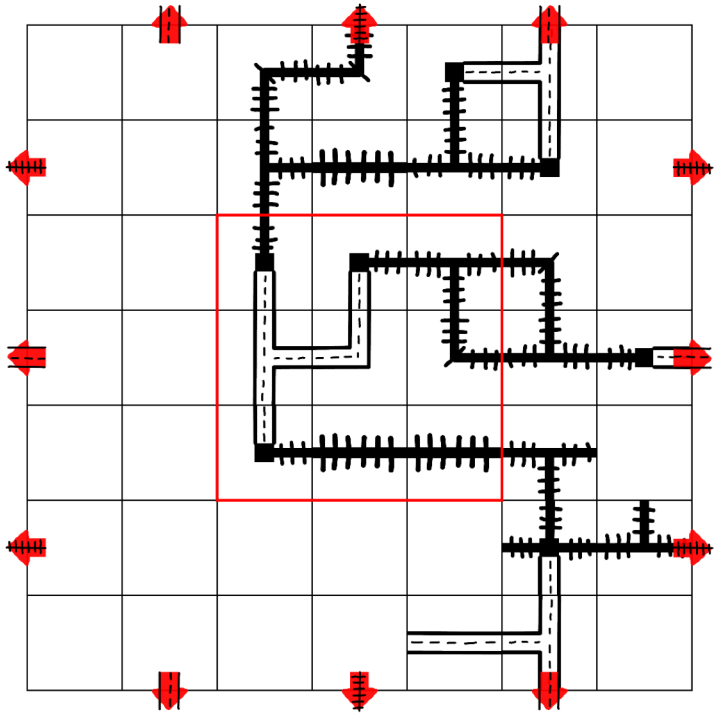
Railway Map

0	0	1	1	1	0	0
0	0	1	1	1	1	0
0	0	1	1	1	1	0
0	0	0	0	1	1	1
0	0	1	1	1	1	0
0	0	0	0	0	1	1
0	0	0	0	0	0	0

Highway map

0	0	0	0	1	1	0
0	0	0	0	0	1	0
0	0	1	1	0	0	0
0	0	1	1	0	0	1
0	0	1	0	0	0	0
0	0	0	0	0	1	0
0	0	0	0	0	0	1

Board Placement



getLongestHighwayCount()

```
public void getLongestHighwayCount()
{
    for (int i = 0;i<7;i++)
    {
        for (int j = 0;j<7;j++)
        {
            if (this.board[i][j] != null)
            {
                int[][] clone = new int[7][7];

                for (int x = 0; x < 7; x++)
                {
                    for (int y = 0; y < 7; y++)
                    {
                        clone[x][y] = this.highwayConnections[x][y];
                    }
                }

                if (this.highwayConnections[i][j] == 1)
                {
                    findHighwayNeighboursRecursively((Tile) this.board[i][j],  count: 1, clone);
                }
            }
        }
    }
}
```

findHighwayNeighboursRecursively(Tile x, int count, int[][] map)

```
public void findHighwayNeighboursRecursively(Tile x, int count, int[][] map)
{
    if (count > longestHighway)
        longestHighway = count;


    int[][] clone = new int[7][7];
    for (int i = 0;i<7;i++)
    {
        for (int j = 0;j<7;j++)
        {
            clone[i][j] = map[i][j];
        }
    }

    clone[x.shape[1]][x.shape[2]] = 0;

    try
    {
        if ((x.checkHighwayConnection((Tile)this.board[x.shape[1]-1][x.shape[2]])) && (map[x.shape[1]-1][x.shape[2]] == 1))
        {
            findHighwayNeighboursRecursively((Tile) this.board[x.shape[1]-1][x.shape[2]],  count: count+1, clone);
        }
    }
    catch (Exception e)
    {

    }
    try
    {
        if ((x.checkHighwayConnection((Tile)this.board[x.shape[1]+1][x.shape[2]])) && (map[x.shape[1]+1][x.shape[2]] == 1))
        {
            findHighwayNeighboursRecursively((Tile) this.board[x.shape[1]+1][x.shape[2]],  count: count+1, clone);
        }
    }
    catch (Exception e)
    {

    }
```

Continued

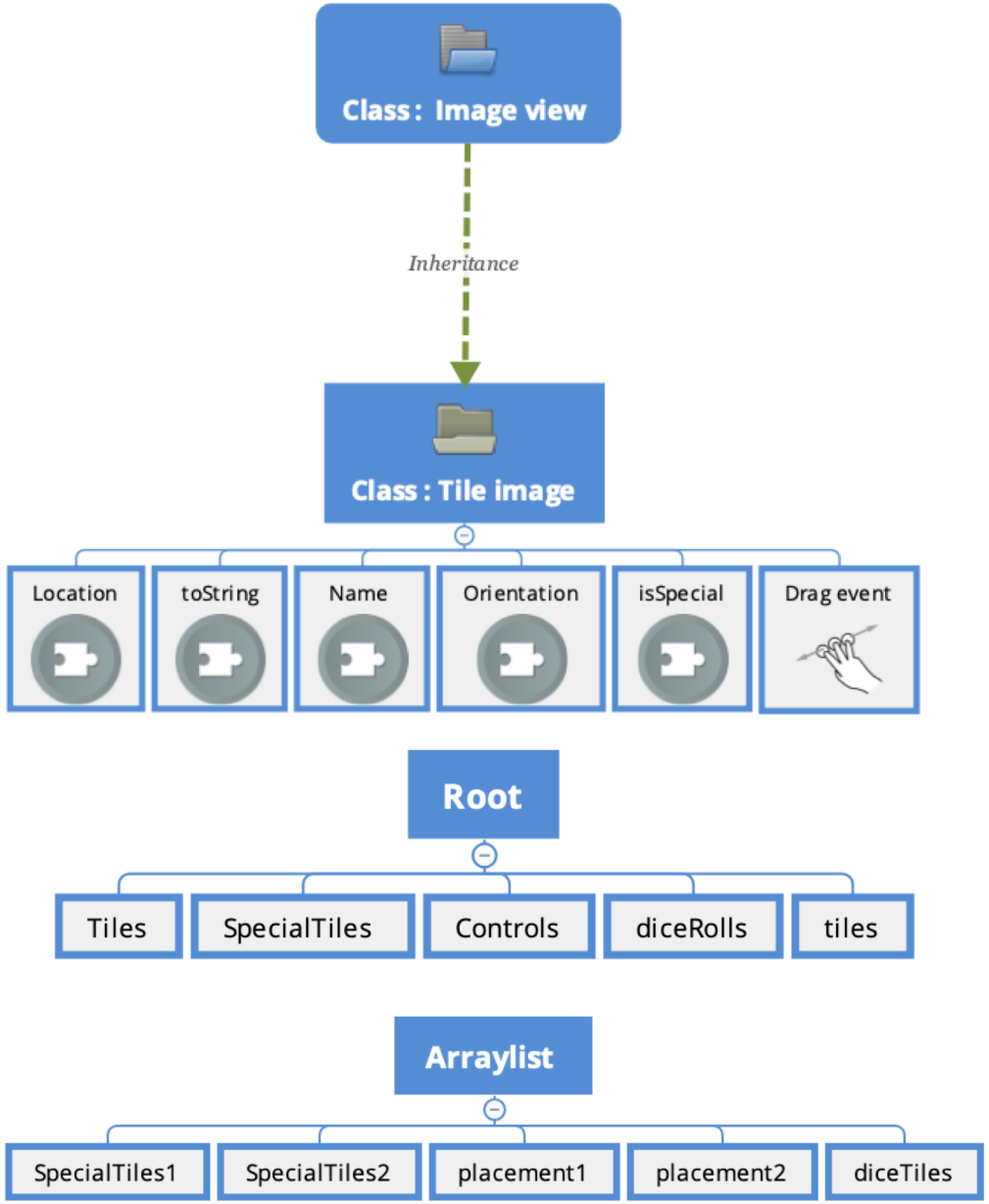
```
        try
        {
            if ((x.checkHighwayConnection((Tile)this.board[x.shape[1]][x.shape[2]-1])) && (map[x.shape[1]][x.shape[2]-1] == 1))
            {
                findHighwayNeighboursRecursively((Tile)this.board[x.shape[1]][x.shape[2]-1], count: count+1, clone);
            }
        }
        catch (Exception e)
        {

        }
        try
        {
            if ((x.checkHighwayConnection((Tile)this.board[x.shape[1]][x.shape[2]+1])) && (map[x.shape[1]][x.shape[2]+1] == 1))
            {
                findHighwayNeighboursRecursively((Tile)this.board[x.shape[1]][x.shape[2]+1], count: count+1, clone);
            }
        }
        catch (Exception e)
        {

        }

    }
```

Computer Opponent

- Generating Valid Moves
  - Recusrsive function goes through every possible tile position for each dice roll in all orders
  - All possible moves are updated in HashSet Container
  - The move which returns highest score is selected as the best move

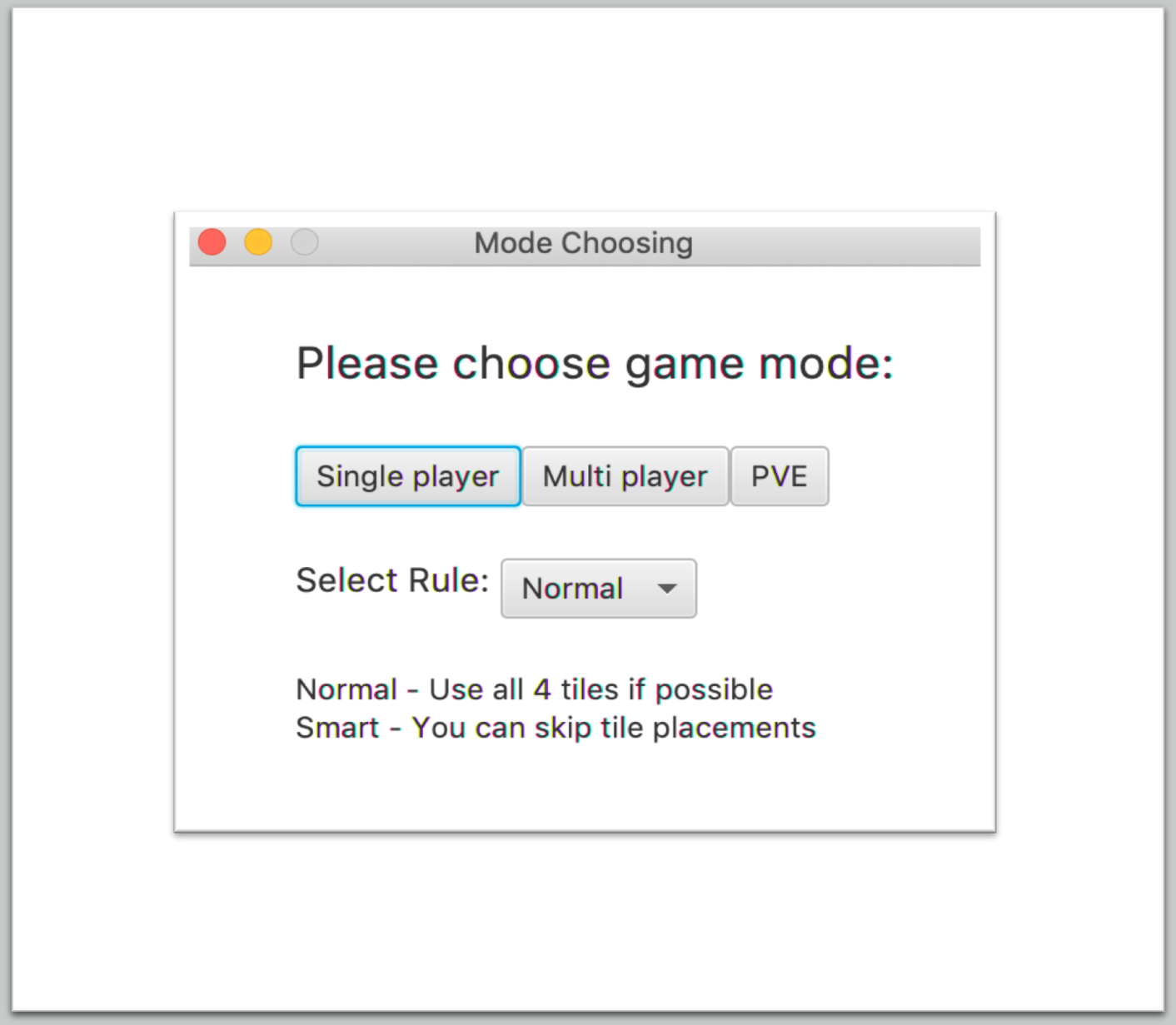
UI Structure

- Tile image class inherits the properties of Tile image
- Drag event enable user to move the tile.
- The Tile image have five properties, which are location, name, orientation, isSpecial and toString respectively.

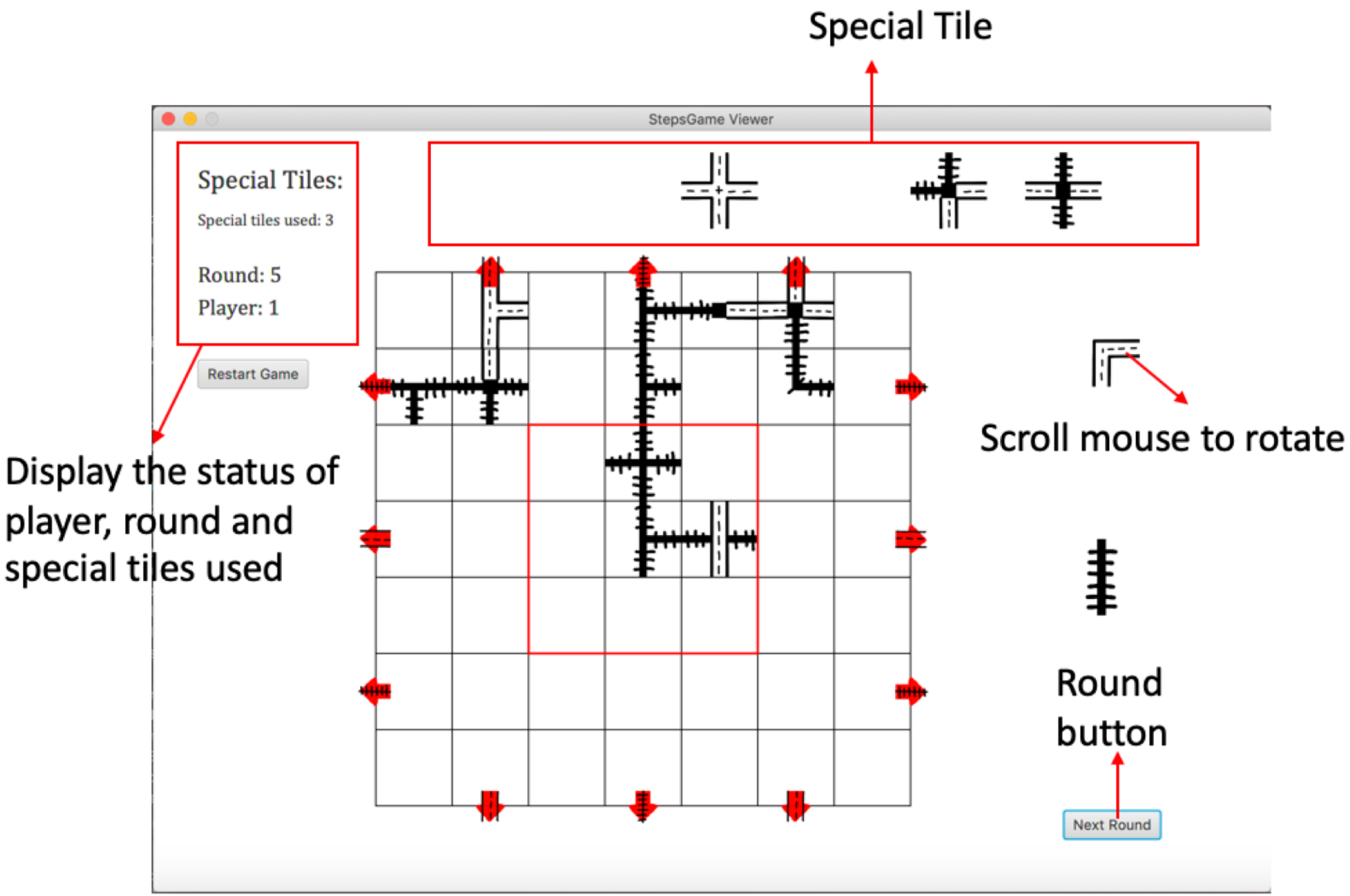


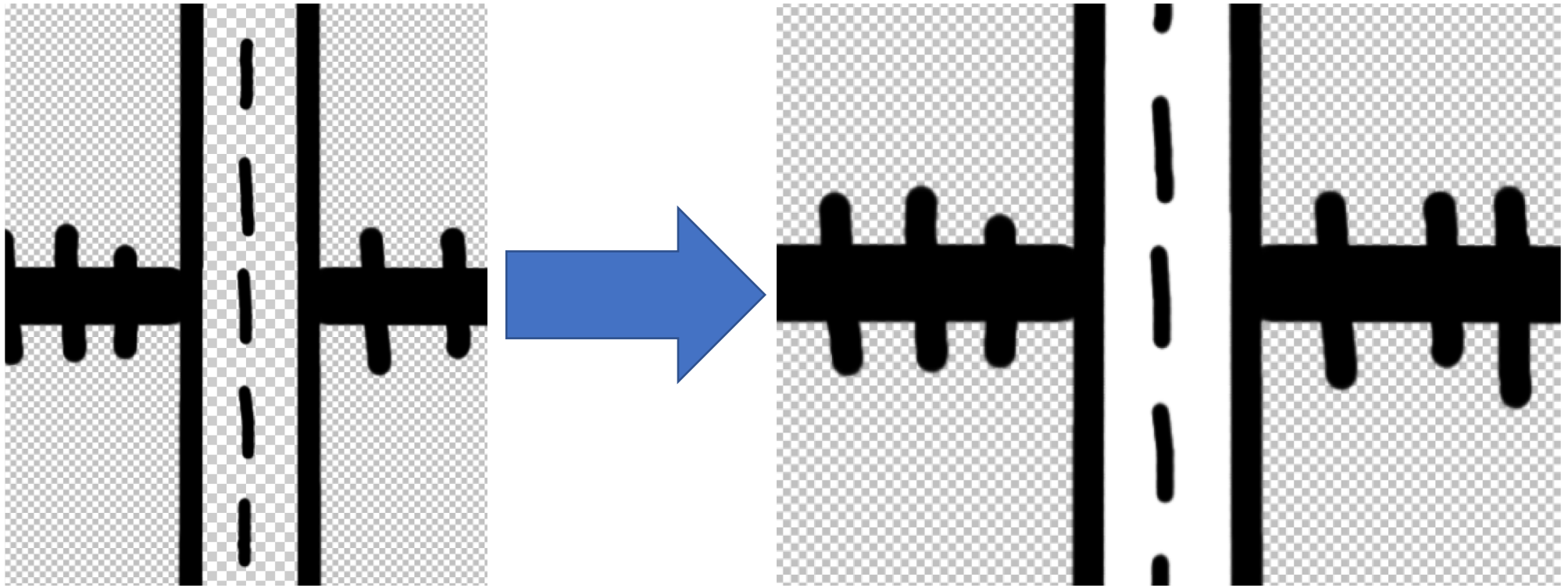
Start Menu

- Single player
- Multi player : People VS People
- PVE : People VS Environment
- Select Rule : Normal and Smart



Game Menu


Special Tile
StepsGame Viewer
Special Tiles:
Special tiles used: 3
Round: 5
Player: 1
Restart Game
Display the status of player, round and special tiles used
Scroll mouse to rotate
Round button
Next Round

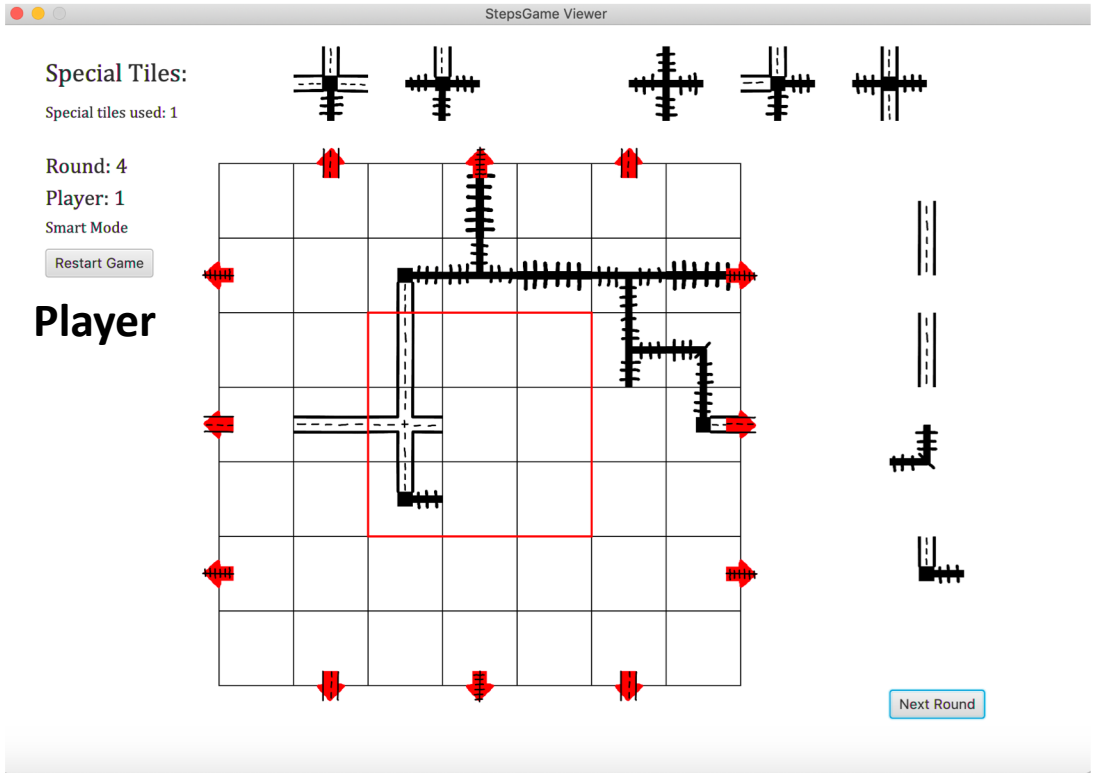


Small changes with the Tile picture

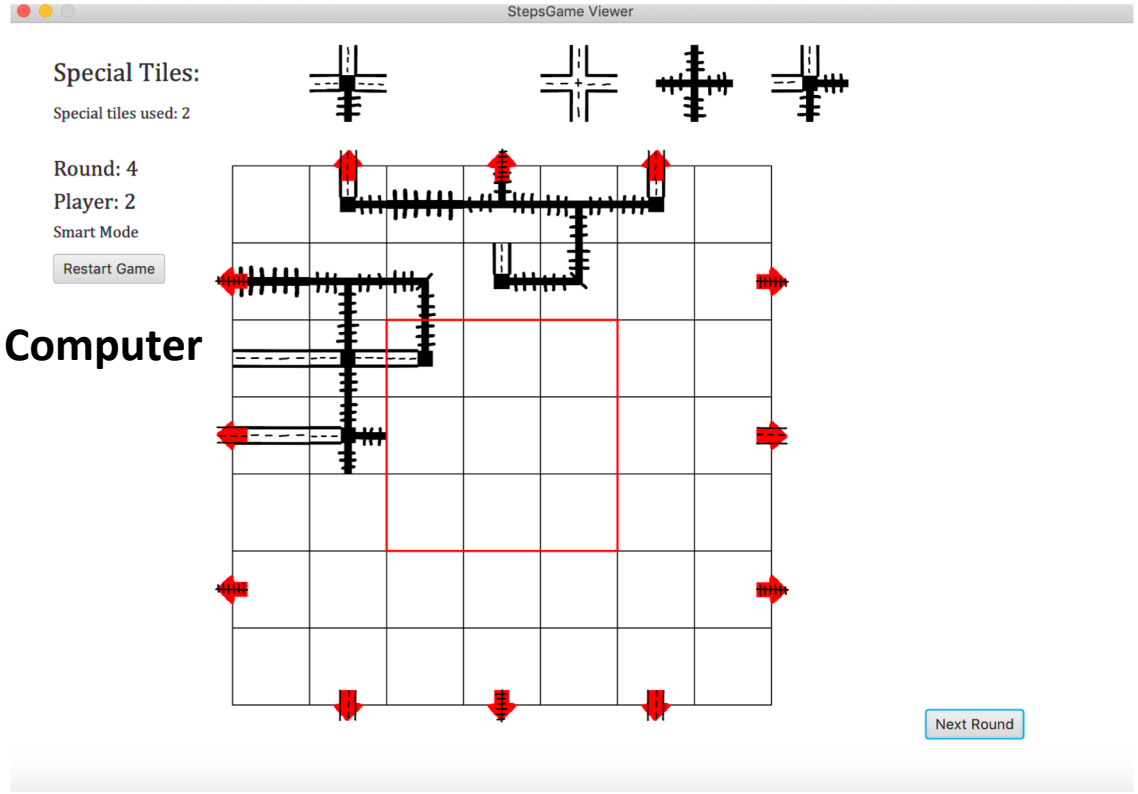
User could drag the Tile precisely

Game Process : Player VS Environment

Round 4 : Player VS Computer



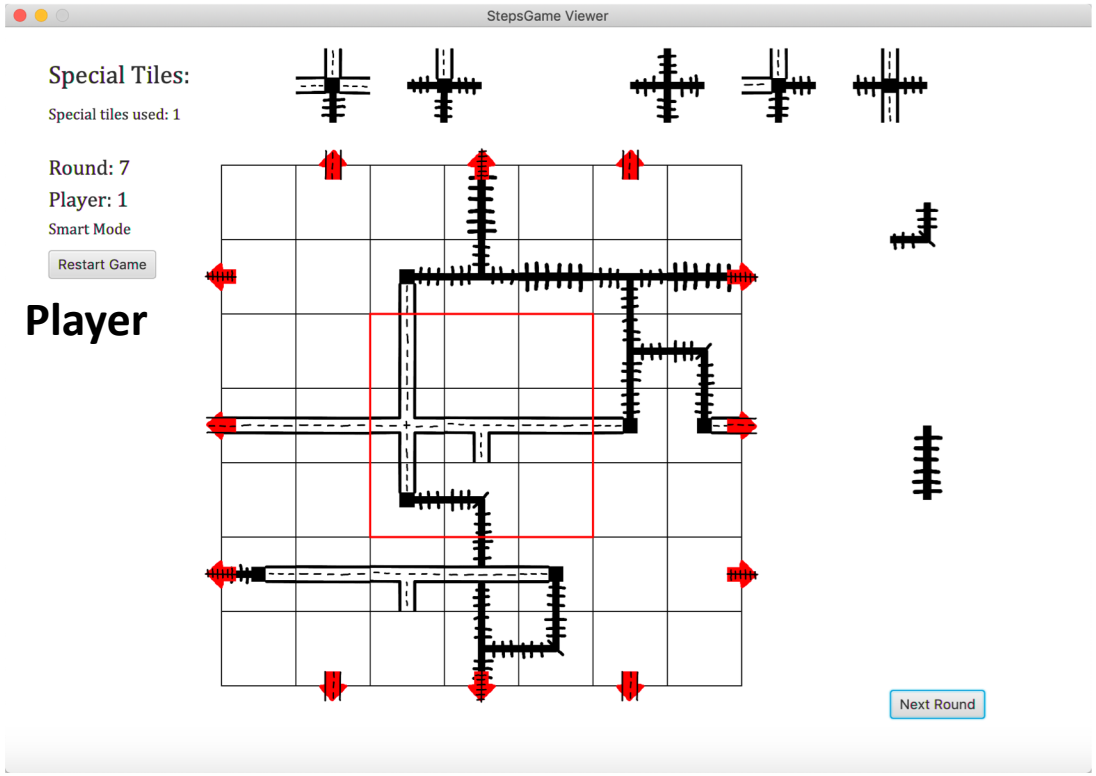
Player



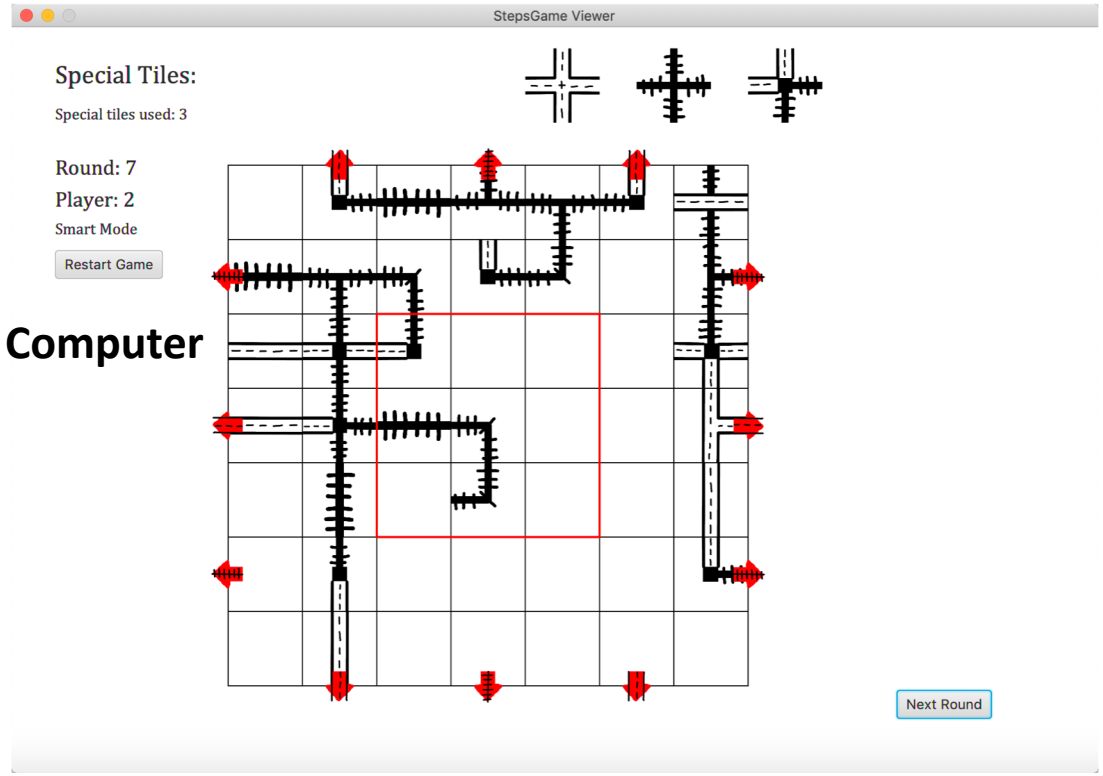
Computer

Game Process : People VS Environment

Round 7 : Player VS Computer

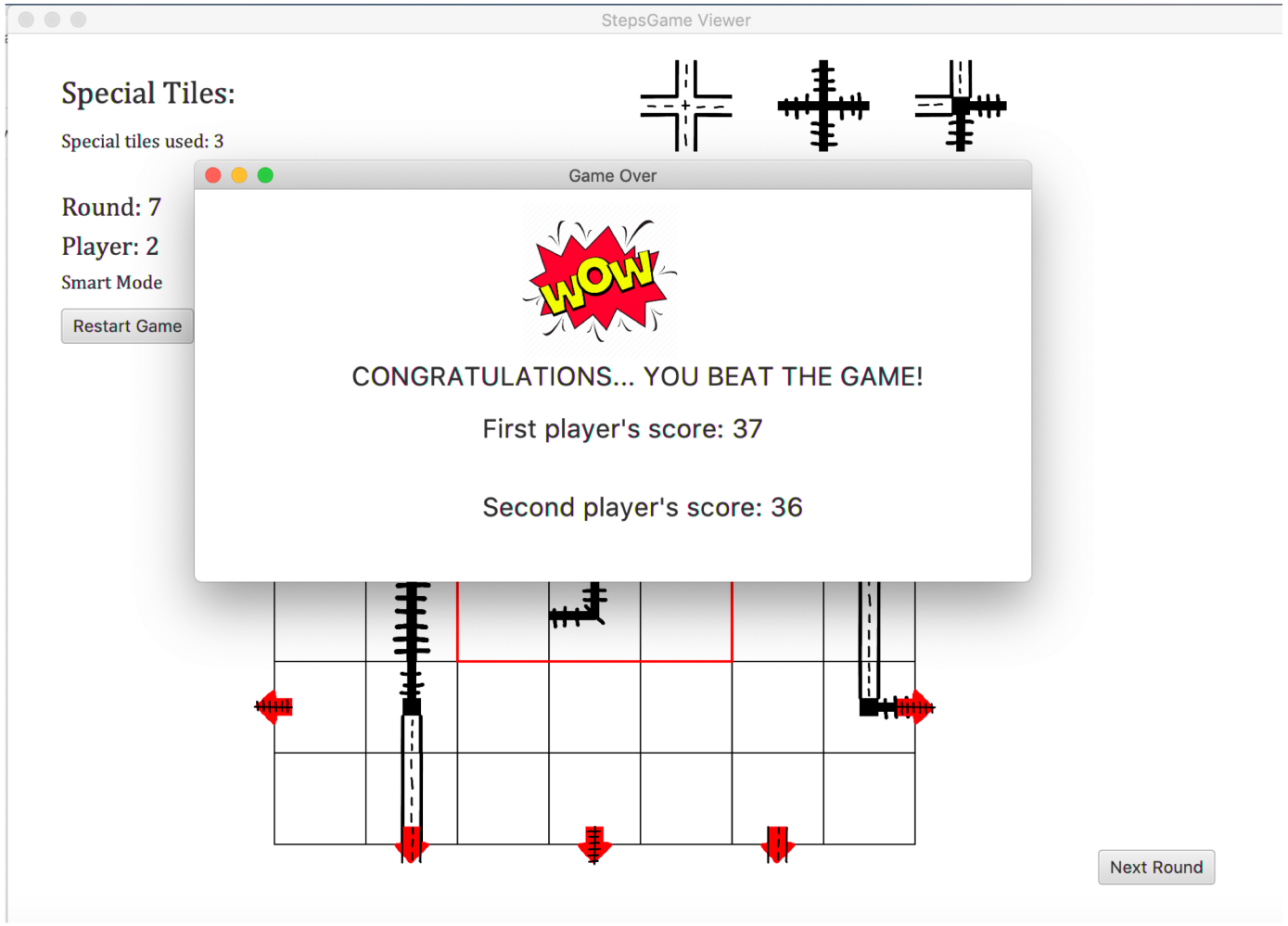


Player

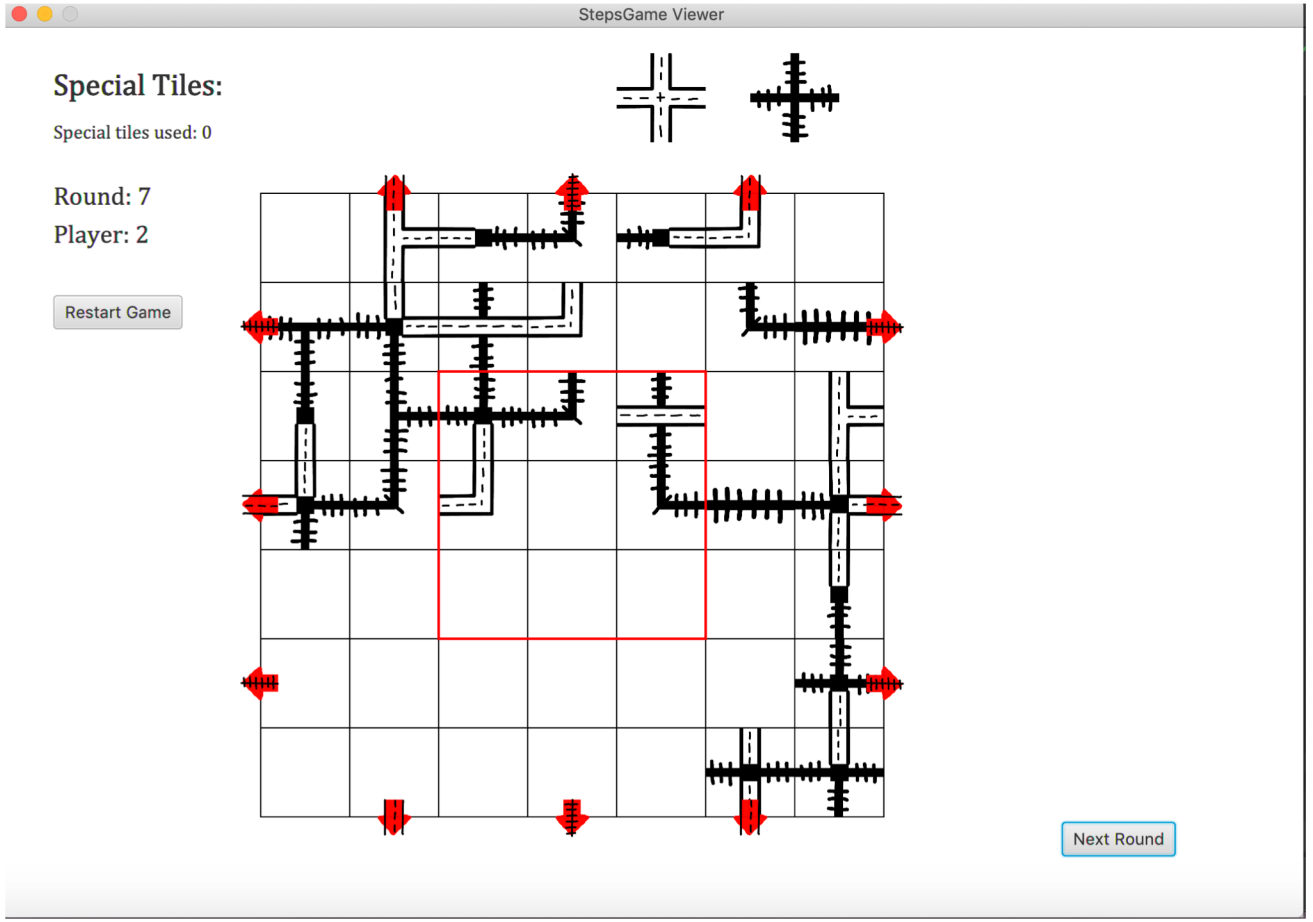


Computer

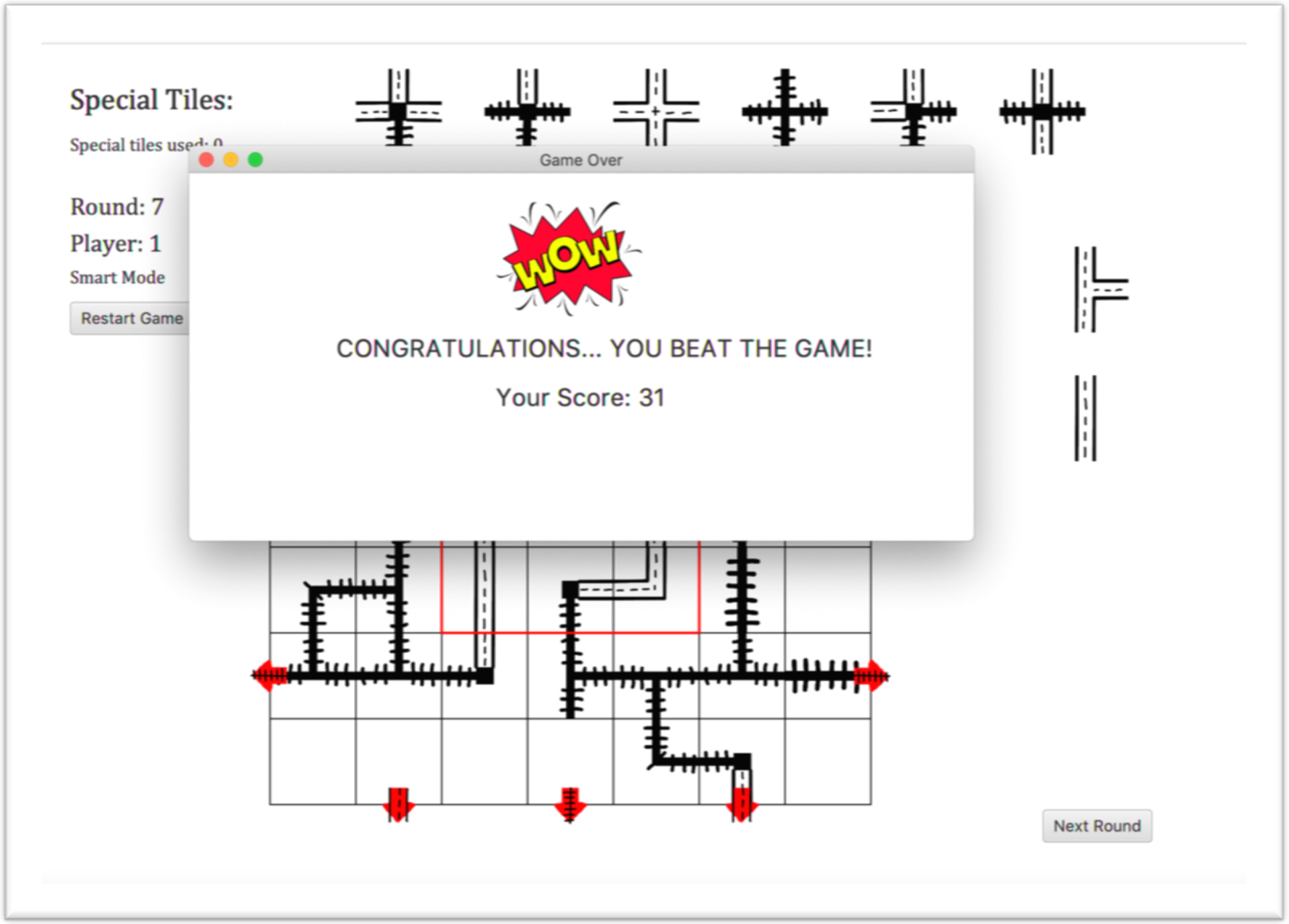
Game Over


StepsGame Viewer
Special Tiles:
Special tiles used: 3
Round: 7
Player: 2
Smart Mode
Restart Game
Game Over
WOW
CONGRATULATIONS... YOU BEAT THE GAME!
First player's score: 37
Second player's score: 36
Next Round

Game Process : Single player



Game Over


Special Tiles:
Special tiles used: 0
Round: 7
Player: 1
Smart Mode
Restart Game
Game Over
WOW
CONGRATULATIONS... YOU BEAT THE GAME!
Your Score: 31
Next Round

Thanks