A Mini Project Report

On

Implementation of Ludo Game

Submitted in partial fulfillment of requirements for the Course
CSE18R272 - JAVA PROGRAMMING

Bachelor's of Technology

In

Computer Science and Engineering

Submitted By

K.Nithya

9918004084

M.Malathy

9918004061

Under the guidance of

Dr. R. RAMALAKSHMI

(Associate Professor)

Department of Computer Science and Engineering

Kalasalingam Academy of Research and Education

Anand Nagar, Krishnankoil-626126

APRIL 2020

ABSTRACT

Ludo Game Stimulation is a computer program that imitates the manual method of playing Ludo Board game. Ludo is a board game that will be played by two to four players. In this game, the players race their four coins/tokens from start to end according to the die rolls.This game is derived from an Indian game named Pachisi. The game and its variations are popular in many countries. The random chance is high. The computerized ludo game is designed using Java Applet. Even one player can play this game by making computer as opponents.

This game has many strategies such as Aggressive,Defensive,Human strategy,Lone pawn,etc.

DECLARATION

I hereby declare that the work presented in this report entitled "**Ludo Game"**", in partial fulfilment of the requirements for the course CSE18R272-Java Programming and submitted in **Department of Computer Science and Engineering, Kalasalingam Academy of Research and Education** (**Deemed to be University**) is an authentic record of our own work carried out during the period from **Jan 2020** under the guidance of Mr. **Dr. R. Ramalakshmi** (Associate Professor).

The work reported in this has not been submitted by me for the award of any other degree of this or any other institute.

K.Nithya
9918004084
M.Malathy
9918004061

ACKNOWLEDGEMENT

First and foremost,I wish to thank the **Almighty God** for his grace and benediction to complete this project work successfully. I would like to convey my special thanks from the bottom of my heart to my dear **Parents** and **Family members** for their honest support for the completion of this project work.

I express my deep gratitude to "Kalvivallal" Thiru **T.Kalasalingam** B.com., Founder Chairman, "Ilayavallal" **Dr.K.Shridharan** Ph.D., Chancellor, **Dr.Shasi Anand** Ph.D., Vice President (Academic),**Mr.S. ArjunKalasalingam** M.S., Vice President (Administration), **Dr.R.Nagaraj** Vice-Chancellor, **Dr. V.Vasudevan** Ph.D., Registrar , **Dr.P.Deepalakshmi** Ph.D., Deen (School of Computing). And also a special thanks to **Dr.A.Francis Saviour Devaraj**, Head of the Department of CSE, Kalasalingam Academy of Research and Education for granting the permission and providing necessary facilities to carry out Project work.I would also like to express my special appreciation and profound thanks to my enthusiastic Project Supervisor **Dr.R.Ramalakshmi** Ph.D, Associate Professor at Kalasalingam Academy of Research and Education [KARE] for her inspiring guidance,constant encouragement with my work during all stages.I am extremely glad that I had a chance to do my Project under my Guide who truly practices and appreciates deep thinking,she game me the moral support and the freedom I needed to move on.

K.Nithya
9918004084
M.Malathy
9918004061

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

The Ludo game is an ancient game which plays important role in the Brain development and helps in reasoning skills,enhance critical thinking as well as boost spatial reasoning. This project is to implement ludo game using JAVA.

1.0.1 Statement of the problem

The following problems are observed from the local ludo game:

1. Local Ludo game involves atleast two persons in the game i.e, it cannot be played by just one person.

2. Sometimes there will be no accuracy in counting like counting 4 moves instead of 5 moves

1.0.2 Objectives

The Objectives of this project are as follows:

1. To develop a code for implementing ludo game

2. To be able to create real life features of ludo game

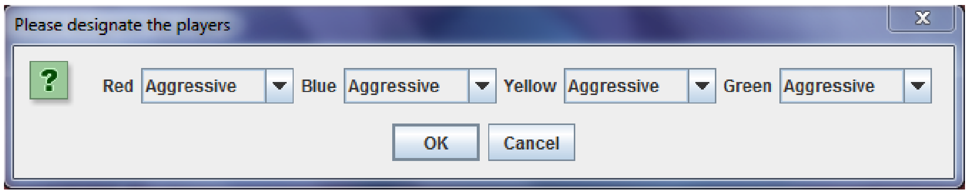
3. To improve the reasoning ability of people

4. To act as a good form of relaxation

Chapter 2

PROJECT DESCRIPTION

The Ludo Game is implemented using JDK and JRE. Many packages are used in this program.Such as, java.io, java.util, java.applet,etc. The java.awt and java.applet is used to create the Ludo board. In this game,player can select their strategy such as Aggressive,Defensive,Lone Pawn,Human Strategy,etc. Maximum 4 player can play this game.


Please designate the players

Red Aggressive Blue Aggressive Yellow Aggressive Green Aggressive

Figure 2.1: Output1

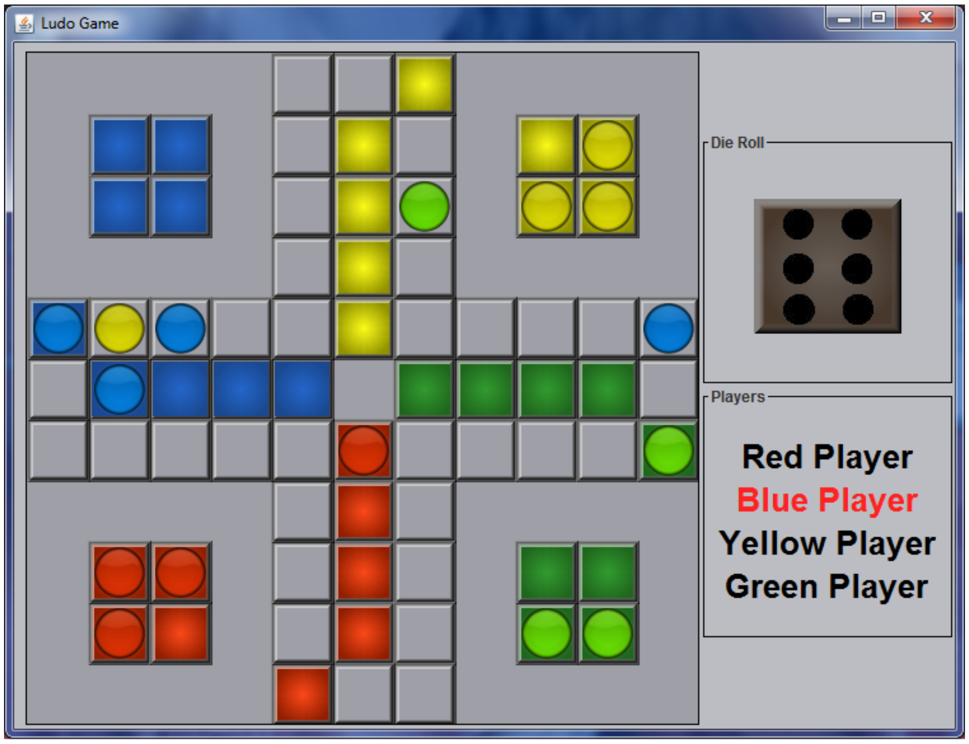


Figure 2.2: Output2

Chapter 3

CONCLUSION

This Project implements the computerized Ludo Game. This is a simple Multiplayer stratergy board game. Some of the problems of Local Ludo game is rectified in this game. The LUDO game is improvised using Java.

The future enhancement may included with sound effects. Eventhough this model has many options,it is a 2-D game. In future, it may be designed with 3-D animation and much more options.

Appendices

SOURCE CODE

```
import java.awt.*;
import java.io.*;
import java.util.*;
import javax.imageio.*;
import javax.swing.*;


public class LudoGame extends JPanel
{
  private static final long serialVersionUID =
     ↪ 4096501410402784791L;
  private final static int BOARDLEFTOFFSET = 9;
  private final static int BOARDTOPOFFSET = 7;
  private final static int GRIDSIZE = 48;
  private final static int GRIDNUM = 11;
  private final static Point[][] THEGRID = new Point[
     ↪ GRIDNUM][GRIDNUM];
  public static int SLEEP = 200;
  private boolean theShowMustGoOn = true;
  private int thePlayer = 0;
  private final JLabel redLabel;
  private final JLabel blueLabel;
  private final JLabel yellowLabel;
  private final JLabel greenLabel;
  private final ArrayList<Pawn> redPawns = new
     ↪ ArrayList<Pawn>();
  private final ArrayList<Pawn> bluePawns = new
     ↪ ArrayList<Pawn>();
  private final ArrayList<Pawn> yellowPawns = new
     ↪ ArrayList<Pawn>();
  private final ArrayList<Pawn> greenPawns = new
     ↪ ArrayList<Pawn>();
  private final JLayeredPane boardPane;
  private final Die theDie;
  private HomeField redHome;
  private HomeField blueHome;
  private HomeField greenHome;
  private HomeField yellowHome;
private final ArrayList<GoalField> redGoal = new
   ↪ ArrayList<GoalField>();
private final ArrayList<GoalField> blueGoal = new
   ↪ ArrayList<GoalField>();
private final ArrayList<GoalField> yellowGoal = new
   ↪ ArrayList<GoalField>();
private final ArrayList<GoalField> greenGoal = new
   ↪ ArrayList<GoalField>();
private final ArrayList<Player> players = new
   ↪ ArrayList<Player>();
public Runnable continueAfterThreadEnd = new Runnable
   ↪ ()
{
 @Override
 public void run()
 {
   continueGameRound();
 }
};
 private LudoGame()
 {
  setLayout(new BoxLayout(this, BoxLayout.PAGE_AXIS))
     ↪ ;
  final ImageIcon boardBackground = createImageIcon("
     ↪ src/board_bkg.png");
  final ImageIcon redPawnImg = createImageIcon("src/
     ↪ red_disk.png");
  final ImageIcon bluePawnImg = createImageIcon("src/
     ↪ blue_disk.png");
  final ImageIcon yellowPawnImg = createImageIcon("
     ↪ src/yellow_disk.png");
  final ImageIcon greenPawnImg = createImageIcon("src
     ↪ /green_disk.png");
  final ImageIcon dieImg = createImageIcon("src/die_1
     ↪ .png");
   setupTheGrid();
   boardPane = new JLayeredPane();
   boardPane.setPreferredSize(new Dimension(540, 540)
      ↪ );
   JLabel board = new JLabel(boardBackground);
```

```
boardPane.add(board, new Integer(0));
Dimension boardSize = board.getPreferredSize();
board.setBounds(BOARDLEFTOFFSET, BOARDTOPOFFSET,
   ↪ boardSize.width, boardSize.height);
JPanel rightPane = new JPanel();
rightPane.setLayout(new GridBagLayout());

JPanel dieLayer = new JPanel();
dieLayer.setPreferredSize(new Dimension(200, 200))
   ↪ ;
dieLayer.setBorder(BorderFactory.
   ↪ createTitledBorder(BorderFactory.
   ↪ createLineBorder(Color.black), "Die␣Roll"));
dieLayer.setLayout(new GridBagLayout());

JLabel die = new JLabel(dieImg);
dieLayer.add(die, new GridBagConstraints());
dieLayer.setBackground(new Color(188, 189, 194));

JPanel playersLayer = new JPanel();
playersLayer.setPreferredSize(new Dimension(200,
   ↪ 200));
playersLayer.setBorder(BorderFactory.
   ↪ createTitledBorder(BorderFactory.
   ↪ createLineBorder(Color.black), "Players"));
redLabel = new JLabel("Red␣Player");
blueLabel = new JLabel("Blue␣Player");
yellowLabel = new JLabel("Yellow␣Player");
greenLabel = new JLabel("Green␣Player");
playersLayer.setLayout(new GridBagLayout());
playersLayer.setBackground(new Color(188, 189,
   ↪ 194));

   GridBagConstraints playGrid = new
      ↪ GridBagConstraints();
   playGrid.gridy = 0;
   playersLayer.add(redLabel, playGrid);
   playGrid.gridy = 1;
   playersLayer.add(blueLabel, playGrid);
   playGrid.gridy = 2;
```

```
    playersLayer.add(yellowLabel, playGrid);
    playGrid.gridy = 3;
    playersLayer.add(greenLabel, playGrid);

    GridBagConstraints theGrid = new
       ↪ GridBagConstraints();
    theGrid.gridy = 0;
    rightPane.add(dieLayer, theGrid);
    theGrid.gridy = 1;
    rightPane.add(playersLayer, theGrid);
    rightPane.setBackground(new Color(188, 189,
       ↪ 194));

    setupTheFields();

    addPawns(redPawnImg, redPawns, redHome);
    addPawns(bluePawnImg, bluePawns, blueHome);
    addPawns(yellowPawnImg, yellowPawns, yellowHome
       ↪ );
    addPawns(greenPawnImg, greenPawns, greenHome);

    setupThePlayers();
    theDie = Die.getInstance(die);
    setLayout(new BoxLayout(this, BoxLayout.X_AXIS)
       ↪ );
    add(boardPane);
    add(rightPane);
  }
  protected void startTheGame()
  {
     startGameRound();
    }
    private void startGameRound()
    {
     if (theShowMustGoOn)
      {
        Player pl = players.get(thePlayer);
        System.out.println("Player␣" + thePlayer +
           ↪ "␣starts␣turn␣...");
        pl.setLabelIsTurn();
        int roll = rollDie();
        sleep(SLEEP);
        pl.doMove(roll);
      }
  }
  protected void continueGameRound()
  {
   sleep(SLEEP);
   if (theDie.lastRoll() == 6)
   {
     startGameRound();
    }
   else
   {
     Player pl = players.get(thePlayer);
     System.out.println("Turn done!\n");
     if (pl.checkIfGoalFull())
      {
          System.err.println("We have a winner!!!
              ↪ ");
          theShowMustGoOn = false;
      }
         pl.setLabelNotTurn();
         sleep(SLEEP);
      if (theShowMustGoOn)
      {
          System.out.println("Round done! Next 
              ↪ round starting...\n");
          thePlayer++;
          if (thePlayer > 3)
          {
            thePlayer = 0;
          }
          startGameRound();
        }
     }
  }
  private int rollDie()
  {
   int playerRoll = theDie.roll();
   System.out.println("Roll:␣" + playerRoll);
   theDie.setImage(createImageIcon("src/die_" +
      ↪ playerRoll + ".png"));
   sleep(SLEEP * 2);
   return playerRoll;
  }
   private void sleep(final long milli)
   {
    try
    {
      Thread.sleep(milli);
     }
   catch (InterruptedException ie)
   {
    System.err.println("Unexpected␣timing␣error.␣
       ↪ Aborting␣thread␣sleep");
   }
  }
  private void setupTheGrid()
  {
   for (int i = 0; i < GRIDNUM; i++)
   {
    for (int j = 0; j < GRIDNUM; j++)
     {
       THEGRID[i][j] = new Point(BOARDLEFTOFFSET
          ↪ + (i * GRIDSIZE),
                      BOARDTOPOFFSET + (j *
                         ↪ GRIDSIZE));
          }
          }
  }
 private void setupTheFields()
 {
  final int[] gridJ =
     ↪ {10,10,9,8,7,6,6,6,6,6,5,4,4,
          4,4,4,3,2,1,0,0,0,1,2,3,4,4,4,4,
          4,5,6,6,6,6,6,7,8,9,10};
  final int[] gridI = {5,4,4,4,4,4,3,2,1,0,0,0,1
      ,2,3,4,4,4,4,4,5,6,6,6,6,6,7,8,9,
           10,10,10,9,8,7,6,6,6,6,6};
```

```
Field lastField = null;
BasicField firstField = null;
for (int i = 0; i < 40; i++)
 {
  BasicField theTrack = new BasicField(THEGRID[
      ↪ gridI[i]][gridJ[i]]);
  if (i % 10 == 0)
  {
   if (i == 0)
   {
     firstField = theTrack;
     int[] goalJ = { 9, 8, 7, 6 };
     int[] goalI = { 5, 5, 5, 5 };
        setupTheGoals(redGoal, goalI, goalJ,
           ↪ theTrack);
    }
    else if (i == 10)
   {
     int[] goalJ = { 5, 5, 5, 5 };
      int[] goalI = { 1, 2, 3, 4 };
      setupTheGoals(blueGoal, goalI, goalJ,
         ↪ theTrack);
    }
   else if (i == 20)
   {
     int[] goalJ = { 1, 2, 3, 4 };
     int[] goalI = { 5, 5, 5, 5 };
        setupTheGoals(yellowGoal, goalI, goalJ,
           ↪  theTrack);
    }
   else if (i == 30)
   {
        int[] goalJ = { 5, 5, 5, 5 };
        int[] goalI = { 9, 8, 7, 6 };
        setupTheGoals(greenGoal, goalI, goalJ,
           ↪ theTrack);
    }
  }
 else if ((i - 1) % 10 == 0)
 {
      if (i == 1)
      {
        int[] homeJ = { 8, 9, 8, 9 };
        int[] homeI = { 1, 1, 2, 2 };
        redHome = setupTheHome(homeI, homeJ,
            ↪ theTrack);
        }
      else if (i == 11)
      {
            int[] homeJ = { 1, 1, 2, 2 };
            int[] homeI = { 2, 1, 2, 1 };
            blueHome = setupTheHome(homeI, homeJ,
                ↪ theTrack);
       }
      else if (i == 21)
      {
        int[] homeJ = { 2, 1, 2, 1 };
        int[] homeI = { 9, 9, 8, 8 };
        yellowHome = setupTheHome(homeI, homeJ,
            ↪ theTrack);
       }
      else if (i == 31) {
      int[] homeJ = { 9, 9, 8, 8 };
      int[] homeI = { 8, 9, 8, 9 };
      greenHome = setupTheHome(homeI, homeJ,
          ↪ theTrack);
      }
    }
    if (lastField != null)
    {
      lastField.setNextField(theTrack);
      }
      lastField = theTrack;
      }
      lastField.setNextField(firstField);
  }
 private void setupTheGoals(final ArrayList<
    ↪ GoalField> theGoal,
      final int[] gridI, final int[] gridJ, final
         ↪  BasicField linker)
  {
   GoalField lastField = null;
   GoalField currentField = null;
   for (int i = 3; i >= 0; i--)
   {
    currentField = new GoalField(THEGRID[gridI[i
       ↪ ]][gridJ[i]]);
    theGoal.add(currentField);
    if (lastField != null)
     {
       currentField.setNextGoalField(lastField);
      }
     lastField = currentField;
   }
   linker.setGoalField(currentField);
   }
 private HomeField setupTheHome(final int[] gridI,
          final int[] gridJ,final BasicField entry
             ↪ )
 {
   final ArrayList<Point> points = new ArrayList<
      ↪ Point>();
   for (int i = 0; i < gridI.length; i++)
   {
     points.add(THEGRID[gridI[i]][gridJ[i]]);
   }
   HomeField hf = new HomeField(points);
   hf.setNextField(entry);
   return hf;
   }
  private void setupThePlayers()
  {
   ArrayList<ArrayList<GoalField>> goalFields =
                                   new ArrayList<
                                      ↪ ArrayList<
                                      ↪ GoalField>>
                              (Arrays.asList(redGoal,
                                 ↪ blueGoal,
                                      yellowGoal,
                                         ↪ greenGoal
```

```
                                            ↪ ));
ArrayList<ArrayList<Pawn>> pawns = new
    ↪ ArrayList<ArrayList<Pawn>>
                                (Arrays.asList(
                                    ↪ redPawns,
                                    ↪ bluePawns,
                                        yellowPawns,
                                            ↪ greenPawns
                                            ↪ ));
HomeField[] homeFields = { redHome, blueHome,
    ↪ yellowHome,greenHome };
JLabel[] playerLabels = { redLabel, blueLabel,
    ↪ yellowLabel,greenLabel };
String[] names = { "Red", "Blue", "Yellow", "
    ↪ Green" };
String[] strategies = { "Aggressive", "
    ↪ Defensive", "Lone␣Pawn",
                                        "Many␣Pawns", "
                                            ↪ Human␣
                                            ↪ Player" };
@SuppressWarnings("rawtypes")
JComboBox[] choices = { new JComboBox<String>(
    ↪ strategies)
                                    ,new JComboBox<
                                        ↪ String>(
                                        ↪ strategies),
                        new JComboBox<String>(
                            ↪ strategies),
                            new JComboBox<String>(
                                ↪ strategies) };
JPanel prompt = new JPanel();
for (int i = 0; i < 4; i++)
{
prompt.add(new JLabel(names[i]));
prompt.add(choices[i]);
 }
int result = JOptionPane.showConfirmDialog(null
    ↪ , prompt,
                "Please␣designate␣the␣players",
                    ↪ JOptionPane.OK_CANCEL_OPTION
                    ↪ );
        if (result == JOptionPane.OK_OPTION)
        {
         for (int i = 0; i < 4; i++)
           {
             Strategy someStrategy;
             switch (choices[i].getSelectedItem().
                ↪ toString())
             {
                case "Aggressive":
                        someStrategy = new
                           ↪ AggressiveStrategy();
                        break;
                case "Lone␣Pawn":
                        someStrategy = new
                           ↪ LonePawnStrategy();
                        break;
                case "Many␣Pawns":
                        someStrategy = new
                           ↪ ManyPawnsStrategy();
                        break;
                case "Defensive":
                        someStrategy = new
                           ↪ DefensiveStrategy();
                        break;
                case "Human␣Player":
                default:
                        someStrategy = new
                           ↪ HumanStrategy();
                        break;
                }
           players.add(new Player(someStrategy,
             ↪ goalFields.get(i),
                    homeFields[i], playerLabels[i],
                       ↪ pawns.get(i), this));
              }
             }
             else
             {
                System.exit(1);
        }
    }
    private void addPawns(final ImageIcon imgSrc,
        ↪ final ArrayList<Pawn>
                                pawnList, final
                                    ↪ HomeField home)
    {
     for (int i = 0; i < 2; i++)
      {
        for (int j = 0; j < 2; j++) {
            JButton jl = new JButton(imgSrc);
             jl.setBorderPainted(false);
             jl.setContentAreaFilled(false);
             boardPane.add(jl, new Integer(1));
             Dimension size = new Dimension(jl.
                 ↪ getIcon().getIconWidth(),
                         jl.getIcon().getIconHeight
                             ↪ ());
             jl.setBounds(0, 0, size.width, size.
                 ↪ height);
             Pawn p = new Pawn(jl, home);
             pawnList.add(p);
              }
             }
    }
    private ImageIcon createImageIcon(final String
        ↪ src)
    {
    try {
        BufferedImage bluePawn = ImageIO.read(new
            ↪ File(src));
        ImageIcon icon = new ImageIcon(bluePawn);
         return icon;
        } catch (IOException ioe) {
        ioe.printStackTrace();
        return null;
        }
    }
    private static void createGUI()
    {
            JFrame frame = new JFrame("Ludo Game");
            frame.setDefaultCloseOperation(JFrame.
                ↪ EXIT_ON_CLOSE);
            LudoGame contentPane = new LudoGame();
            contentPane.setOpaque(true);
            contentPane.setBackground(new Color(188, 189,
                ↪ 194));
              frame.setContentPane(contentPane);

                  frame.pack();
                  frame.setVisible(true);
                  contentPane.startTheGame();
          }
          public static void main(String[] args)
          {
                  createGUI();
          }
}
```

```
import java.awt.*;
import java.util.*;
public class HomeField extends Field
 {
          private final ArrayList<Pawn> homePawns = new
              ↪ ArrayList<Pawn>();
          private final ArrayList<Point> thePoints = new
              ↪ ArrayList<Point>();
          public HomeField(final ArrayList<Point> points)
          {
                  super(points.get(0));
                  for (Point p : points)
                  {
                          thePoints.add(p);
                  }
          }
          @Override
          public final Pawn getPawn()
          {
                  Pawn p = null;
                  if (hasPawn())
```

```
                {
                        p = homePawns.remove(0);
                }
                return p;
        }
        public final Pawn peekAtPawn()
        {
                return homePawns.get(0);
        }
        @Override
        public final boolean hasPawn()
        {
                return (homePawns.size() > 0);
        }
        public final int getPawnCount()
        {
                return homePawns.size();
        }
        public final boolean isFull()
        {
                return (homePawns.size() == 4);
        }
        public final void setPawns(final ArrayList<Pawn
           ↪ > pawns)
        {
                for (Pawn p : pawns)
                {
                        setPawn(p);
                }
        }

        @Override
        public final void setPawn(final Pawn pawn)
         {
                if (pawn != null)
                {
                        homePawns.add(0, pawn);
                }
        }
        @Override
        public Point getPoint() {
                return getPoints().get(homePawns.size()
                   ↪ );
        }

        public ArrayList<Point> getPoints()
        {
                return thePoints;
        }
}
```

```
import java.awt.*;
public class GoalField extends Field
 {
        public GoalField(final Point point)
        {
                super(point);
        }
        public final void setNextGoalField(final
           ↪ GoalField goalField)
        {
                setNextField(goalField);
        }
}
```

```
import java.awt.event.*;
import java.util.*;

public class HumanStrategy implements Strategy,
   ↪ MouseListener
 {
    private Player thePlayer = null;
    private int theRoll = 0;
    private ArrayList<Pawn> validPawns;
    private ArrayList<Move> validMoves;
    @Override
    public void chooseMove(final Player player, final
       ↪ int dieRoll)
    {
```

```
validPawns = new ArrayList<Pawn>();
validMoves = new ArrayList<Move>();
this.thePlayer = player;
this.theRoll = dieRoll;
for (Pawn p : thePlayer.getPawns())
{
  p.getImgSrc().addMouseListener(this);
  if (p.isAtHome()&& thePlayer.checkValidMove(
     ↪ thePlayer.getHomeField().getNextField()
     ↪ ) && theRoll == 6)
  {
    validPawns.add(p);
    validMoves.add(new Move(thePlayer.
       ↪ getHomeField().peekAtPawn(),thePlayer
       ↪ .getHomeField().getNextField()));
  }
  else if (p.isAtGoal())
  {
    Field f = thePlayer.checkMovePawnGoal(p,(
       ↪ GoalField) p.getField(), theRoll);
     if (f != null)
     {
        validPawns.add(p);
        validMoves.add(new Move(p, f));
      }
    }
   else if (p.isAtBasic())
   {
        Field f = thePlayer.checkMovePawnBasic(
           ↪ p,(BasicField) p.getField(),
           ↪ dieRoll);
        if (f != null)
        {
          validPawns.add(p);
          validMoves.add(new Move(p, f));
        }
    }
}
System.out.println("Valid␣human␣pawns?␣" +
   ↪ validPawns.size());
```

```
}
  @Override
  public void mouseClicked(MouseEvent e) {}
  @Override
  public void mouseEntered(MouseEvent e) {}
  @Override
  public void mouseExited(MouseEvent e) {}
  @Override
  public void mousePressed(MouseEvent e)
   {
     if (e.getButton() == MouseEvent.BUTTON1)
     {
       System.out.println("Process␣human␣move␣
          ↪ attempt...");
       Pawn thePawn = null;
       Object clickSource = e.getSource();
       for (Pawn p : thePlayer.getPawns())
       {
        if (p.getImgSrc() == clickSource)
         {
          thePawn = p;
          break;
          }
        }
     for (Pawn p : validPawns)
     {
      if (thePawn == p)
       {
          if (thePawn.isAtHome())
            {
             thePlayer.getHomeField().getPawn();
             }
          sendMoveToPlayer(thePlayer,validMoves.
             ↪ get(validPawns.indexOf(p)));
          for (Pawn q : thePlayer.getPawns())
          {
            while (q.getImgSrc().
               ↪ getMouseListeners().length > 0)
            {
             q.getImgSrc().removeMouseListener(q.
                    ↪ getImgSrc().getMouseListeners
                    ↪ ()[0]);
                  }
                }
                break;
              }
            }
            if (validMoves.size() < 1)
            {
             System.err.println("No␣valid␣moves␣exist!")
                ↪ ;
             sendMoveToPlayer(thePlayer, null);
             for (Pawn q : thePlayer.getPawns())
             {
                 while (q.getImgSrc().getMouseListeners
                    ↪ ().length > 0)
                  {
                   q.getImgSrc().removeMouseListener(q.
                      ↪ getImgSrc().getMouseListeners()
                      ↪ [0]);
                 }
              }
             }
        }
      }
      @Override
      public void mouseReleased(MouseEvent e) {}
      @Override
      public void sendMoveToPlayer(final Player player,
         ↪  final Move move)
      {
        player.takeMove(move);
      }
}
```

```
import java.util.*;
public class LonePawnStrategy implements Strategy
{
        private HomeField theHome;
        private ArrayList<Pawn> thePawns;
```

```
@Override
public void chooseMove(final Player player,
   ↪ final int dieRoll)
{
        theHome = player.getHomeField();
        thePawns = player.getPawns();

        ArrayList<Pawn> basicFieldPawns = new
           ↪ ArrayList<Pawn>();
        Field field = theHome.getNextField();
        do
        {
                if (field.hasPawn())
                {
                        if (thePawns.contains(
                           ↪ field.getPawn()))
                        {
                                basicFieldPawns
                                   ↪ .add(0,
                                   ↪ field.
                                   ↪ getPawn()
                                   ↪ );
                        }
                }
                field = field.getNextField();
        } while (field != theHome.getNextField
           ↪ ());
        for (Pawn p : basicFieldPawns)
        {
                Field f = player.
                   ↪ checkMovePawnBasic(p, (
                   ↪ BasicField) p.getField(),
                   ↪ dieRoll);
                if (f != null)
                {
                        sendMoveToPlayer(player
                           ↪ , new Move(p, f))
                           ↪ ;
                        return;
                }
                }

                for (Pawn p : thePawns)
                {
                        if (p.isAtGoal())
                        {
                                Field f = player.
                                   ↪ checkMovePawnGoal
                                   ↪ (p, (GoalField) p
                                   ↪ .getField(),
                                   ↪ dieRoll);
                                if (f != null)
                                {
                                        sendMoveToPlayer
                                           ↪ (player,
                                           ↪ new Move(
                                           ↪ p, f));
                                        return;
                                }
                        }
                }

                sendMoveToPlayer(player, null);
        }
        @Override
        public void sendMoveToPlayer(final Player
           ↪ player, final Move move)
        {
                player.takeMove(move);
        }
}
```

```
import java.util.*;
public class ManyPawnsStrategy implements Strategy
{
        private HomeField theHome;
        private ArrayList<Pawn> thePawns;
        @Override
        public void chooseMove(final Player player,
           ↪ final int dieRoll)
        {
                theHome = player.getHomeField();
                thePawns = player.getPawns();
                ArrayList<Pawn> chosen = new ArrayList<
                   ↪ Pawn>();
                Field field = theHome.getNextField();
                do
                {
                        if (field.hasPawn())
                        {
                                if (thePawns.contains(
                                   ↪ field.getPawn()))
                                {
                                        chosen.add(
                                           ↪ field.
                                           ↪ getPawn()
                                           ↪ );
                                }
                        }
                        field = field.getNextField();
                } while (field != theHome.getNextField
                   ↪ ());
                for (Pawn p : chosen)
                {
                        Field f = player.
                           ↪ checkMovePawnBasic(p, (
                           ↪ BasicField) p.getField(),
                           ↪ dieRoll);
                        if (f != null)
                        {
                                sendMoveToPlayer(player
                                   ↪ , new Move(p, f))
                                   ↪ ;
                                return;
                        }
                }
                for (Pawn p : thePawns)
                {
                        if (p.isAtGoal())
                        {
                                Field f = player.
                                    ↪ checkMovePawnGoal
                                    ↪ (p, (GoalField) p
                                    ↪ .getField(),
                                    ↪ dieRoll);
                                if (f != null)
                                {
                                        sendMoveToPlayer
                                            ↪ (player,
                                            ↪ new Move(
                                            ↪ p, f));
                                        return;
                                }
                        }
                }
                sendMoveToPlayer(player, null);
        }
        @Override
        public void sendMoveToPlayer(final Player
            ↪ player, final Move move)
        {
                player.takeMove(move);
        }

}
```

```
public class Move
{
        private final Pawn thePawn;
        private final Field theField;

        public Move(final Pawn pawn, final Field field)
        {
                this.thePawn = pawn;
                this.theField = field;
        }

        public final Pawn getPawn()
        {
                return thePawn;
        }

        public final Field getField()
        {
                return theField;
        }
}
```

```
import java.awt.*;
import javax.swing.*;
public class Pawn
{
        private final JButton imgSrc;
        private Field location;
        private HomeField homeLoc;
        private int pos = 0;
        public Pawn(final JButton source, final
           ↪ HomeField loc)
        {
                this.imgSrc = source;
                this.homeLoc = loc;
                moveToField(loc);
        }
        protected JButton getImgSrc()
        {
                return imgSrc;
        }
        private void setPosition(final Point pos)
        {
                imgSrc.setLocation(pos);
        }
        public final Field getField()
        {
                return location;
        }
        public final boolean isAtHome()
        {
                return (pos == 0 ? true : false);
        }
        public final boolean isAtBasic()
```

```
	{
		return (pos == 1 ? true : false);
	}
	public final boolean isAtGoal()
	{
		return (pos == 2 ? true : false);
	}
	public final void moveToHome()
	{
		moveToField(homeLoc);
	}
	public final void moveToField(final Field field
	   ↪ )
	{
		if (field != homeLoc)
		{
			location.setPawn(null);
			if (field.hasPawn())
			{
				field.getPawn().
				   ↪ moveToHome();
			}
		}
		this.location = field;
		setPosition(location.getPoint());
		location.setPawn(this);
		if (field.getClass() == BasicField.
		   ↪ class)
		{
			this.pos = 1;
		}
		else if (field.getClass() == GoalField.
		   ↪ class)
		{
			this.pos = 2;
		}
		else if (field.getClass() == HomeField.
		   ↪ class)
		{
			this.pos = 0;
```

```
                }
                else
                {
                        System.err.println("Pawn unable
                           ↪  to identify current 
                           ↪ field type");
                }
        }
}
```

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
public class Player
{
  private final Strategy strategy;
  private final ArrayList<GoalField> goalField;
  private final HomeField homeField;
  private final JLabel playLabel;
  private final LudoGame parent;
  private final ArrayList<Pawn> pawns;
  public Player(final Strategy strategy,final ArrayList
     ↪ <GoalField> goalField,
                final HomeField homeField,final JLabel
                   ↪ playLabel,
                final ArrayList<Pawn> pawns,final
                   ↪ LudoGame parent)
  {
    this.strategy = strategy;
   this.goalField = goalField;
    this.homeField = homeField;
    this.playLabel = playLabel;
   this.playLabel.setFont(new Font("Sans-Serif", Font.
      ↪ BOLD, 26));
   this.pawns = pawns;
   this.parent = parent;
   }
    public final void doMove(final int dieRoll)
    {
       if (dieRoll == 6 && homeField.getPawnCount() > 0
                && checkMovePawnHome() != null)
      {
         takeMove(new Move(homeField.getPawn(),
            ↪ homeField.getNextField()));
       }
    else
    {
       strategy.chooseMove(this, dieRoll);
     }
}
public void takeMove(final Move move)
{
     if (move != null)
      {
         Pawn p = move.getPawn();
          Field f = move.getField();

         while (move.getField() != f)
         {
               if (f.getClass() == BasicField.class)
                {
                  if (((BasicField) f).hasGoalField())
                   {
                        if (((BasicField) f).
                           ↪ getGoalField() ==
                           ↪ goalField.get(3))
                         {
                           f = ((BasicField) f).
                              ↪ getGoalField();
                          }
                        else {
                        f = f.getNextField();
                        }
                    }
                    else {
                     f = f.getNextField();
                     }
                  }
                else {
                  f = f.getNextField();
                }
              }
              p.moveToField(f);
            }
            SwingUtilities.invokeLater(parent.
               ↪ continueAfterThreadEnd);
        }
    public final HomeField getHomeField()
    {
        return homeField;
     }
    public final GoalField getEntryGoalField()
    {
        return goalField.get(goalField.size() - 1);
     }
     public final ArrayList<Pawn> getPawns()
     {
        return pawns;
      }
       public Field checkMovePawnHome()
       {
        if (checkValidMove(homeField.getNextField()))
         {
            return homeField.getNextField();
         }
        else {
        return null;
        }
       }
  public Field checkMovePawnBasic(final Pawn pawn,
     ↪ final BasicField field ,final int distance)
  {
      if (distance == 0 && checkValidMove(field))
       {
        return field;
        }
      else if (distance == 0 && !checkValidMove(field))
         ↪  {
        return null;
      }
    else
      {
        if (field.hasGoalField()) {
        if (field.getGoalField() == goalField.get(3))
            ↪  {
            return checkMovePawnGoal(pawn, goalField.
                ↪ get(3),distance - 1);
            }
          else {
           System.out.println("Noticed␣a␣goal␣field␣
               ↪ ...␣failed␣to␣be␣interested");
           return checkMovePawnBasic(pawn,(BasicField)
               ↪  field.getNextField(), distance - 1);
          }
        }
        else {
          return checkMovePawnBasic(pawn,(BasicField)
              ↪ field.getNextField(), distance - 1);
           }
         }
       }
  public Field checkMovePawnGoal(final Pawn pawn,
     ↪ final GoalField goal,final int distance)
   {
      if (distance == 0) {
       if (checkValidMove(goal)) {
              return goal;
      }
      else {
        return null;
       }
      } else if (!goal.hasNextField()) {
                     return null;
        } else {
              return checkMovePawnGoal(pawn, (
                  ↪ GoalField) goal.getNextField(),
                  ↪ distance - 1);
         }
      }
   public boolean checkIfGoalFull() {
    boolean isFull = true;
    for (GoalField g : goalField) {
     isFull &= g.hasPawn();
      }
    return isFull;
    }
public boolean checkIfGoalOccupied()
 {
    boolean hasPawn = false;
    for (GoalField g : goalField) {
    hasPawn |= g.hasPawn();
 }
    return hasPawn;
}
public int getGoalOccupiedCount() {
    int numPawns = 0;
    for (GoalField g : goalField) {
            if (g.hasPawn()) {
              numPawns++;
            }
          }
            return numPawns;
    }
    public void setLabelNotTurn() {
            playLabel.setForeground(new Color(0
               ↪ x000000));
    }
    public void setLabelIsTurn() {
            playLabel.setForeground(new Color(0
               ↪ xff2222));
    }
    public boolean checkValidMove(final Field field
       ↪ ) {
            if (field.hasPawn()) {
                    if (isOwnPawn(field.getPawn()))
                       ↪ {
                            System.out.println("
                               ↪ Invalid␣move␣
                               ↪ considered,␣own␣
                               ↪ pawn␣at␣field␣
```

```
                                        ↪ location");
                                    return false;
                            } else {
                                    return true;
                            }
                    } else {
                      return true;
                    }
        }
        private boolean isOwnPawn(final Pawn foundPawn)
        {
         boolean ownPawn = false;
         for (Pawn p : pawns)
          {
            ownPawn |= (p == foundPawn);
          }
          return ownPawn;
        }
}
```

```
public interface Strategy
{
        public void chooseMove(final Player player,
            ↪ final int dieRoll);
        public void sendMoveToPlayer(final Player
            ↪ player, final Move move);
 }
```

```
import java.awt.*;
public class BasicField extends Field
{
        private GoalField goalLink;

        public BasicField(final Point point)
        {
                super(point);
                setGoalField(null);
        }

        public Pawn removePawn()
        {
                Pawn p = getPawn();
                setPawn(null);
                return p;
        }

        public final boolean hasGoalField()
        {
                return (getGoalField() != null ? true :
                   ↪  false);
        }

        public final GoalField getGoalField()
        {
                return goalLink;
        }

        public final void setGoalField(final GoalField
           ↪ goal)
         {
                this.goalLink = goal;
        }
}
```

```
import java.util.*;
public class AggressiveStrategy implements Strategy
{
        private HomeField theHome;
        private ArrayList<Pawn> thePawns;
        @Override
        public void chooseMove(final Player player,
           ↪ final int dieRoll)
         {
            theHome = player.getHomeField();
            thePawns = player.getPawns();

            Pawn frontMostValid = null;
            ArrayList<Pawn> basicFieldPawns = new
               ↪ ArrayList<Pawn>();
            Field field = theHome.getNextField();
```

```
do
{
    if (field.hasPawn())
    {
      if (thePawns.contains(field.getPawn()
         ↪ ))
      {
        basicFieldPawns.add(0, field.
           ↪ getPawn());
      }
    }
  field = field.getNextField();
} while (field != theHome.getNextField());
for (Pawn p : basicFieldPawns)
{
    Field f = player.checkMovePawnBasic(p,
        ↪ (BasicField) p.getField(),dieRoll
        ↪ );
    if (f != null)
    {
      if (frontMostValid == null)
      {
       frontMostValid = p;
      }
     if (f.getPawn() != null)
     {
       sendMoveToPlayer(player, new Move(p,
          ↪  f));
       return;
      }
    }
 }
if (frontMostValid != null)
{
  Field f = player.checkMovePawnBasic(
     ↪ frontMostValid,(BasicField)
     ↪ frontMostValid.getField(), dieRoll)
     ↪ ;
  sendMoveToPlayer(player, new Move(
     ↪ frontMostValid, f));
```

```
                return;
               }
              for (Pawn p : thePawns)
              {
               if (p.isAtGoal())
               {
                  Field f = player.checkMovePawnGoal(p, (
                     ↪ GoalField) p.getField(),dieRoll);
                  if (f != null)
                  {
                   sendMoveToPlayer(player, new Move(p, f
                     ↪ ));
                   return;
                  }
                }
               }
          sendMoveToPlayer(player, null);
         }
      @Override
      public void sendMoveToPlayer(final Player player,
         ↪ final Move move)
      {
         player.takeMove(move);
      }

}
```

```
import java.util.*;
public class DefensiveStrategy implements Strategy
{
   private HomeField theHome;
   private ArrayList<Pawn> thePawns;
   @Override
   public void chooseMove(final Player player, final
      ↪ int dieRoll)
   {
    theHome = player.getHomeField();
    thePawns = player.getPawns();
    Pawn frontMostValid = null;
    ArrayList<Pawn> basicFieldPawns = new ArrayList<
```

```
        ↪ Pawn>();
    Field field = theHome.getNextField();
    do
    {
      if (field.hasPawn())
      {
        if (thePawns.contains(field.getPawn()))
        {
          basicFieldPawns.add(0, field.getPawn());
        }
      }
        field = field.getNextField();
    } while (field != theHome.getNextField());
  ArrayList<Move> rejects = new ArrayList<Move>();
  Move chosen = null;
  for (Pawn p : basicFieldPawns)
  {
    Field f = player.checkMovePawnBasic(p, (BasicField
        ↪ ) p.getField(),dieRoll);
        if (f != null)
        {
          if (frontMostValid == null)
           {
             frontMostValid = p;
             if (player.checkMovePawnBasic(
                 ↪ frontMostValid,(BasicField)
                 ↪ frontMostValid.getField(), dieRoll).
                 ↪ getClass() == GoalField.class)
             {
                 sendMoveToPlayer(player, new Move(
                     ↪ frontMostValid, f));
                 return;
             }
           }

           Field nextField = p.getField();
           for (int i = 0; i < dieRoll; i++)
           {
             if (nextField.getNextField().getPawn() ==
                 ↪ null&& nextField.getNextField().
```

```
                ↪ equals(f))
            {
                chosen = new Move(p, f);
                break;
            }
            else if (nextField.getNextField().getPawn
                ↪ () == null)
            {
                nextField = nextField.getNextField();
            }
            else
            {
                nextField = nextField.getNextField();
                rejects.add(new Move(p, f));
                continue;
             }
      }
      if (chosen != null)
      {
        sendMoveToPlayer(player, chosen);
        return;
       }
      }
     }
     for (Pawn p : thePawns)
     {
      if (p.isAtGoal())
      {
        Field f = player.checkMovePawnGoal(p, (
            ↪ GoalField) p.getField(),dieRoll);
        if (f != null)
        {
         sendMoveToPlayer(player, new Move(p, f));
         return;
         }
       }
      }
      if (rejects.size() != 0)
      {
        sendMoveToPlayer(player, rejects.get(0));
```

```
            return;
          }
          sendMoveToPlayer(player, null);
      }
      @Override
      public void sendMoveToPlayer(final Player player,
         ↪ final Move move)
      {
        player.takeMove(move);
       }
}
```

```
import javax.swing.*;
public class Die
{
        private static Die instance = new Die();
        private int value = 0;
        private static JLabel imgSrc;
        private Die(){}
        public static Die getInstance(final JLabel img)
        {
                if (instance == null)
                {
                        synchronized (Die.class)
                        {
                                if (instance == null)

                                 {
                                        instance = new
                                           ↪ Die();
                                }
                        }
                }
                imgSrc = img;
                return instance;
        }

        public int roll()
       {
                value = (int) Math.ceil(Math.random() *
                ↪  6);
                return value;
        }
        public int lastRoll()
        {
                return value;
        }
        public void setImage(final ImageIcon img)
        {
                imgSrc.setIcon(img);
        }
}
```

```
import java.awt.*;
public class Field
{
        protected Field nextField;
        protected Pawn occupyingPawn;
        protected Point thePoint;
        public Field(final Point point)
        {
                setNextField(null);
                setPawn(null);
                setPoint(point);
        }
        public boolean hasNextField()
        {
                return (getNextField() != null ? true :
                    ↪  false);
        }
        public Field getNextField()
        {
                return nextField;
        }
        protected void setNextField(final Field
           ↪ nextField)
        {
                this.nextField = nextField;
        }
        public boolean hasPawn()
        {
                return (getPawn() != null ? true :
                   ↪ false);
        }
        public Pawn getPawn()
        {
                return occupyingPawn;
        }
        protected void setPawn(final Pawn pawn)
        {
                this.occupyingPawn = pawn;
        }
        public Point getPoint()
        {
                return thePoint;
        }
        protected void setPoint(final Point point)
        {
                this.thePoint = point;
        }
}
```