Contents

[Domino.java 1](#_Toc63430186)

[Location.java 1](#_Toc63430187)

[MultiLingualStringTable.java 1](#_Toc63430188)

[PictureFrame.java 1](#_Toc63430189)

[Barking.java 1](#_Toc63430190)

[ConnectionGenius.java 1](#_Toc63430191)

[IOLibrary.java 1](#_Toc63430192)

[IOSpecialist.java 1](#_Toc63430193)

[Main.java 1](#_Toc63430194)

[SpacePlace.java 1](#_Toc63430195)

# Domino.java

1 package pusey020;

2 /\*\*

3 \* @author Kevan Buckley, maintained by Daniel Pusey

4 \* @version 3.3, 2021

5 \*/

6

7 public class Domino implements Comparable<Domino> {

8 public int high;

9 public int low;

10 public int hx;

11 public int hy;

12 public int lx;

13 public int ly;

14 public boolean placed = false;

15

16 public Domino(int high, int low) {

17 super();

18 this.high = high;

19 this.low = low;

20 }

21

22 public void place(int hx, int hy, int lx, int ly) {

23 this.hx = hx;

24 this.hy = hy;

25 this.lx = lx;

26 this.ly = ly;

27 placed = true;

28 }

29

30 public String toString() {

31 StringBuffer result = new StringBuffer();

32 result.append("[");

33 result.append(Integer.toString(high));

34 result.append(Integer.toString(low));

35 result.append("]");

36 if(!placed){

37 result.append("unplaced");

38 } else {

39 result.append("(");

40 result.append(Integer.toString(hx+1));

41 result.append(",");

42 result.append(Integer.toString(hy+1));

43 result.append(")");

44 result.append("(");

45 result.append(Integer.toString(lx+1));

46 result.append(",");

47 result.append(Integer.toString(ly+1));

48 result.append(")");

49 }

50 return result.toString();

51 }

52

53 /\*\* turn the domino around 180 degrees clockwise\*/

54

55 public void invert() {

56 int tx = hx;

57 hx = lx;

58 lx = tx;

59

60 int ty = hy;

61 hy = ly;

62 ly = ty;

63 }

64

65 public boolean ishl() {

66 return hy==ly;

67 }

68

69

70 public int compareTo(Domino arg0) {

71 if(this.high < arg0.high){

72 return 1;

73 }

74 return this.low - arg0.low;

75 }

76

77

78

79 }

# Location.java

1 package pusey020;

2 import java.awt.Color;

3 import java.awt.Graphics;

4 import java.io.BufferedReader;

5 import java.io.InputStreamReader;

6

7 /\*\*

8 \* @author Kevan Buckley, maintained by Daniel Pusey

9 \* @version 3.3, 2021

10 \*/

11

12 public class Location extends SpacePlace {

13 public int c;

14 public int r;

15 public DIRECTION d;

16 public int tmp;

17 public enum DIRECTION {VERTICAL, HORIZONTAL};

18

19 public Location(int r, int c) {

20 this.r = r;

21 this.c = c;

22 }

23

24 public Location(int r, int c, DIRECTION d) {

25 this(r,c);

26 this.d=d;

27 }

28

29 public String toString() {

30 if(d==null){

31 tmp = c + 1;

32 return "(" + (tmp) + "," + (r+1) + ")";

33 } else {

34 tmp = c + 1;

35 return "(" + (tmp) + "," + (r+1) + "," + d + ")";

36 }

37 }

38

39 public void drawGridLines(Graphics g) {

40 g.setColor(Color.LIGHT\_GRAY);

41 for (tmp = 0; tmp <= 7; tmp++) {

42 g.drawLine(20, 20 + tmp \* 20, 180, 20 + tmp \* 20);

43 }

44 for (int see = 0; see <= 8; see++) {

45 g.drawLine(20 + see \* 20, 20, 20 + see \* 20, 160);

46 }

47 }

48

49 public static int getInt() {

50 BufferedReader r = new BufferedReader(new InputStreamReader(System.in));

51 do {

52 try {

53 return Integer.parseInt(r.readLine());

54 } catch (Exception e) {

55 }

56 } while (true);

57 }

58 }

# MultiLingualStringTable.java

1 package pusey020;

2 /\*\*

3 \* @author Kevan Buckley, maintained by Daniel Pusey

4 \* @version 3.3, 2021

5 \*/

6

7 public class MultiLingualStringTable {

8 private enum LanguageSetting {English, Klingon}

9 private static LanguageSetting cl = LanguageSetting.English;

10 private static String [] em = {"Enter your name:", "Welcome", "Have a good time playing Abominodo"};

11 private static String [] km = {"'el lIj pong:", "nuqneH", "QaQ poH Abominodo"};

12

13 public static String getMessage(int index){

14 if(cl == LanguageSetting.English ){

15 return em[index];

16 } else {

17 return km[index];

18 }

19

20 }

21 }

# PictureFrame.java

1 package pusey020;

2 import java.awt.\*;

3

4 import javax.swing.\*;

5 /\*\*

6 \* @author Kevan Buckley, maintained by Daniel Pusey

7 \* @version 3.3, 2021

8 \*/

9

10 public class PictureFrame {

11 public int[] reroll = null;

12 public Main master = null;

13

14 class DominoPanel extends JPanel {

15 private static final long serialVersionUID = 4190229282411119364L;

16

17 public void drawGrid(Graphics g) {

18 for (int are = 0; are < 7; are++) {

19 for (int see = 0; see < 8; see++) {

20 drawDigitGivenCentre(g, 30 + see \* 20, 30 + are \* 20, 20,

21 master.grid[are][see]);

22 }

23 }

24 }

25

26

27

28 public void drawHeadings(Graphics g) {

29 for (int are = 0; are < 7; are++) {

30 fillDigitGivenCentre(g, 10, 30 + are \* 20, 20, are+1);

31 }

32

33 for (int see = 0; see < 8; see++) {

34 fillDigitGivenCentre(g, 30 + see \* 20, 10, 20, see+1);

35 }

36 }

37

38 public void drawDomino(Graphics g, Domino d) {

39 if (d.placed) {

40 int y = Math.min(d.ly, d.hy);

41 int x = Math.min(d.lx, d.hx);

42 int w = Math.abs(d.lx - d.hx) + 1;

43 int h = Math.abs(d.ly - d.hy) + 1;

44 g.setColor(Color.WHITE);

45 g.fillRect(20 + x \* 20, 20 + y \* 20, w \* 20, h \* 20);

46 g.setColor(Color.RED);

47 g.drawRect(20 + x \* 20, 20 + y \* 20, w \* 20, h \* 20);

48 drawDigitGivenCentre(g, 30 + d.hx \* 20, 30 + d.hy \* 20, 20, d.high,

49 Color.BLUE);

50 drawDigitGivenCentre(g, 30 + d.lx \* 20, 30 + d.ly \* 20, 20, d.low,

51 Color.BLUE);

52 }

53 }

54

55 void drawDigitGivenCentre(Graphics g, int x, int y, int diameter, int n) {

56 int radius = diameter / 2;

57 g.setColor(Color.BLACK);

58 // g.drawOval(x - radius, y - radius, diameter, diameter);

59 FontMetrics fm = g.getFontMetrics();

60 // convert the string to an integer

61 String txt = Integer.toString(n);

62 g.drawString(txt, x - fm.stringWidth(txt) / 2, y + fm.getMaxAscent() / 2);

63 }

64

65 void drawDigitGivenCentre(Graphics g, int x, int y, int diameter, int n,

66 Color c) {

67 int radius = diameter / 2;

68 g.setColor(c);

69 // g.drawOval(x - radius, y - radius, diameter, diameter);

70 FontMetrics fm = g.getFontMetrics();

71 String txt = Integer.toString(n);

72 g.drawString(txt, x - fm.stringWidth(txt) / 2, y + fm.getMaxAscent() / 2);

73 }

74

75 void fillDigitGivenCentre(Graphics g, int x, int y, int diameter, int n) {

76 int radius = diameter / 2;

77 g.setColor(Color.GREEN);

78 g.fillOval(x - radius, y - radius, diameter, diameter);

79 g.setColor(Color.BLACK);

80 g.drawOval(x - radius, y - radius, diameter, diameter);

81 FontMetrics fm = g.getFontMetrics();

82 String txt = Integer.toString(n);

83 g.drawString(txt, x - fm.stringWidth(txt) / 2, y + fm.getMaxAscent() / 2);

84 }

85

86 protected void paintComponent(Graphics g) {

87 g.setColor(Color.YELLOW);

88 g.fillRect(0, 0, getWidth(), getHeight());

89

90 // numbaz(g);

91 //

92 // if (master!=null && master.orig != null) {

93 // drawRoll(g, master.orig);

94 // }

95 // if (reroll != null) {

96 // drawReroll(g, reroll);

97 // }

98 //

99 // drawGrid(g);

100 Location l = new Location(1,2);

101

102 if (master.mode == 1) {

103 l.drawGridLines(g);

104 drawHeadings(g);

105 drawGrid(g);

106 master.drawGuesses(g);

107 }

108 if (master.mode == 0) {

109 l.drawGridLines(g);

110 drawHeadings(g);

111 drawGrid(g);

112 master.drawDominoes(g);

113 }

114 }

115

116 public Dimension getPreferredSize() {

117 // the application window always prefers to be 202x182

118 return new Dimension(202, 182);

119 }

120 }

121

122 public DominoPanel dp;

123

124 public void PictureFrame(Main sf) {

125 master = sf;

126 if (dp == null) {

127 JFrame f = new JFrame("Abominodo");

128 dp = new DominoPanel();

129 f.setContentPane(dp);

130 f.pack();

131 f.setDefaultCloseOperation(JFrame.DO\_NOTHING\_ON\_CLOSE);

132 f.setVisible(true);

133 }

134 }

135

136 public void reset() {

137 // TODO Auto-generated method stub

138

139 }

140

141 }

# Barking.java

1 package pusey020;

2 public class Barking {

3 static transient String[] parrot = {

4 "Ye shall know the truth, and the truth shall make you mad.","Aldous Huxley","",

5 "Experience is a dear teacher, but fools will learn at no other.","Benjamin Franklin","",

6 "If you tell the truth, you don't have to remember anything.","Mark Twain","",

7 "A desire to resist oppression is implanted in the nature of man.","Tacitus","",

8 "The man who carries a cat by the tail learns something that can be learned in no other way.","Mark Twain","",

9 "Truth is confirmed by inspection and delay; falsehood by haste and uncertainty.","Tacitus","",

10 "The man who views the world at 50 the same as he did at 20 has wasted 30 years of his life.","Muhammad Ali","",

11 "Platitudes? Yes, there are platitudes. Platitudes are there because they are true.","Margaret Thatcher","",

12 "The great masses of the people will more easily fall victims to a big lie than to a small one.","Adolf Hitler","",

13 "The essence of a self-reliant and autonomous culture is an unshakeable egoism.","Henry Mencken",""};

14 }

# ConnectionGenius.java

1 package pusey020;

2 import java.net.InetAddress;

3 /\*\*

4 \* @author Kevan Buckley, maintained by Daniel Pusey

5 \* @version 3.3, 2021

6 \*/

7

8 public class ConnectionGenius {

9

10 InetAddress ipa;

11

12 public ConnectionGenius(InetAddress ipa) {

13 this.ipa = ipa;

14 }

15

16 public void fireUpGame() {

17 downloadWebVersion();

18 connectToWebService();

19 awayWeGo();

20 }

21

22 public void downloadWebVersion(){

23 System.out.println("Getting specialised web version.");

24 System.out.println("Wait a couple of moments");

25 }

26

27 public void connectToWebService() {

28 System.out.println("Connecting");

29 }

30

31 public void awayWeGo(){

32 System.out.println("Ready to play");

33 }

34

35 }

14 }

# IOLibrary.java

1 package pusey020;

2 import java.io.\*;

3 import java.net.\*;

4 /\*\*

5 \* @author Kevan Buckley, maintained by Daniel Pusey

6 \* @version 3.3, 2021

7 \*/

8

9 public final class IOLibrary {

10 public static String getString() {

11 BufferedReader r = new BufferedReader(new InputStreamReader(System.in));

12 do {

13 try {

14 return r.readLine();

15 } catch (Exception e) {

16 }

17 } while (true);

18 }

19

20 public static InetAddress getIPAddress() {

21 BufferedReader r = new BufferedReader(new InputStreamReader(System.in));

22 do {

23 try {

24 String[] chunks = r.readLine().split("\\.");

25 byte[] data = { Byte.parseByte(chunks[0]),Byte.parseByte(chunks[1]),Byte.parseByte(chunks[2]),Byte.parseByte(chunks[3])};

26 return Inet4Address.getByAddress(data);

27 } catch (Exception e) {

28 }

29 } while (true);

30 }

31

32 }

# IOSpecialist.java

1 package pusey020;

2

3 /\*\*

4 \* @author Kevan Buckley, maintained by Daniel Pusey

5 \* @version 3.3, 2021

6 \*/

7

8 public class IOSpecialist {

9 public IOSpecialist() {

10 }

11 public String getString(){

12 return IOLibrary.getString();

13 }

14 }

# Main.java

1 package pusey020;

2 import java.awt.Dimension;

3 import java.awt.Graphics;

4 import java.io.\*;

5 import java.net.InetAddress;

6 import java.text.DateFormat;

7 import java.util.\*;

8

9 import javax.swing.JEditorPane;

10 import javax.swing.JFrame;

11 import javax.swing.JScrollPane;

12

13 /\*\*

14 \* If lost and subsequently found please inform K dot A dot Buckley at wlv.ac.uk

15 \*

16 \* @author Kevan Buckley, maintained by Daniel Pusey

17 \* @version 3.3, 2021

18 \*/

19

20 public class Main {

21

22 private String playerName;

23 public List<Domino> \_d;

24 public List<Domino> \_g;

25 public int[][] grid = new int[7][8];

26 public int[][] gg = new int[7][8];

27 int mode = -1;

28 int cf;

29 int score;

30 long startTime;

31

32 PictureFrame pf = new PictureFrame();

33

34 private void generateDominoes() {

35 \_d = new LinkedList<Domino>();

36 int count = 0;

37 int x = 0;

38 int y = 0;

39 for (int l = 0; l <= 6; l++) {

40 for (int h = l; h <= 6; h++) {

41 Domino d = new Domino(h, l);

42 \_d.add(d);

43 d.place(x, y, x + 1, y);

44 count++;

45 x += 2;

46 if (x > 6) {

47 x = 0;

48 y++;

49 }

50 }

51 }

52 if (count != 28) {

53 System.out.println("something went wrong generating dominoes");

54 System.exit(0);

55 }

56 }

57

58 private void generateGuesses() {

59 \_g = new LinkedList<Domino>();

60 int count = 0;

61 int x = 0;

62 int y = 0;

63 for (int l = 0; l <= 6; l++) {

64 for (int h = l; h <= 6; h++) {

65 Domino d = new Domino(h, l);

66 \_g.add(d);

67 count++;

68 }

69 }

70 if (count != 28) {

71 System.out.println("something went wrong generating dominoes");

72 System.exit(0);

73 }

74 }

75

76 void collateGrid() {

77 for (Domino d : \_d) {

78 if (!d.placed) {

79 grid[d.hy][d.hx] = 9;

80 grid[d.ly][d.lx] = 9;

81 } else {

82 grid[d.hy][d.hx] = d.high;

83 grid[d.ly][d.lx] = d.low;

84 }

85 }

86 }

87

88 void collateGuessGrid() {

89 for (int r = 0; r < 7; r++) {

90 for (int c = 0; c < 8; c++) {

91 gg[r][c] = 9;

92 }

93 }

94 for (Domino d : \_g) {

95 if (d.placed) {

96 gg[d.hy][d.hx] = d.high;

97 gg[d.ly][d.lx] = d.low;

98 }

99 }

100 }

101

102 int pg() {

103 for (int are = 0; are < 7; are++) {

104 for (int see = 0; see < 8; see++) {

105 if (grid[are][see] != 9) {

106 System.out.printf("%d", grid[are][see]);

107 } else {

108 System.out.print(".");

109 }

110 }

111 System.out.println();

112 }

113 return 11;

114 }

115

116 int printGuessGrid() {

117 for (int are = 0; are < 7; are++) {

118 for (int see = 0; see < 8; see++) {

119 if (gg[are][see] != 9) {

120 System.out.printf("%d", gg[are][see]);

121 } else {

122 System.out.print(".");

123 }

124 }

125 System.out.println();

126 }

127 return 11;

128 }

129

130 private void shuffleDominoesOrder() {

131 List<Domino> shuffled = new LinkedList<Domino>();

132

133 while (\_d.size() > 0) {

134 int n = (int) (Math.random() \* \_d.size());

135 shuffled.add(\_d.get(n));

136 \_d.remove(n);

137 }

138

139 \_d = shuffled;

140 }

141

142 private void invertSomeDominoes() {

143 for (Domino d : \_d) {

144 if (Math.random() > 0.5) {

145 d.invert();

146 }

147 }

148 }

149

150 private void placeDominoes() {

151 int x = 0;

152 int y = 0;

153 int count = 0;

154 for (Domino d : \_d) {

155 count++;

156 d.place(x, y, x + 1, y);

157 x += 2;

158 if (x > 6) {

159 x = 0;

160 y++;

161 }

162 }

163 if (count != 28) {

164 System.out.println("something went wrong generating dominoes");

165 System.exit(0);

166 }

167 }

168

169 private void rotateDominoes() {

170 // for (Domino d : dominoes) {

171 // if (Math.random() > 0.5) {

172 // System.out.println("rotating " + d);

173 // }

174 // }

175 for (int x = 0; x < 7; x++) {

176 for (int y = 0; y < 6; y++) {

177

178 tryToRotateDominoAt(x, y);

179 }

180 }

181 }

182

183 private void tryToRotateDominoAt(int x, int y) {

184 Domino d = findDominoAt(x, y);

185 if (thisIsTopLeftOfDomino(x, y, d)) {

186 if (d.ishl()) {

187 boolean weFancyARotation = Math.random() < 0.5;

188 if (weFancyARotation) {

189 if (theCellBelowIsTopLeftOfHorizontalDomino(x, y)) {

190 Domino e = findDominoAt(x, y + 1);

191 e.hx = x;

192 e.lx = x;

193 d.hx = x + 1;

194 d.lx = x + 1;

195 e.ly = y + 1;

196 e.hy = y;

197 d.ly = y + 1;

198 d.hy = y;

199 }

200 }

201 } else {

202 boolean weFancyARotation = Math.random() < 0.5;

203 if (weFancyARotation) {

204 if (theCellToTheRightIsTopLeftOfVerticalDomino(x, y)) {

205 Domino e = findDominoAt(x + 1, y);

206 e.hx = x;

207 e.lx = x + 1;

208 d.hx = x;

209 d.lx = x + 1;

210 e.ly = y + 1;

211 e.hy = y + 1;

212 d.ly = y;

213 d.hy = y;

214 }

215 }

216

217 }

218 }

219 }

220

221 private boolean theCellToTheRightIsTopLeftOfVerticalDomino(int x, int y) {

222 Domino e = findDominoAt(x + 1, y);

223 return thisIsTopLeftOfDomino(x + 1, y, e) && !e.ishl();

224 }

225

226 private boolean theCellBelowIsTopLeftOfHorizontalDomino(int x, int y) {

227 Domino e = findDominoAt(x, y + 1);

228 return thisIsTopLeftOfDomino(x, y + 1, e) && e.ishl();

229 }

230

231 private boolean thisIsTopLeftOfDomino(int x, int y, Domino d) {

232 return (x == Math.min(d.lx, d.hx)) && (y == Math.min(d.ly, d.hy));

233 }

234

235 private Domino findDominoAt(int x, int y) {

236 for (Domino d : \_d) {

237 if ((d.lx == x && d.ly == y) || (d.hx == x && d.hy == y)) {

238 return d;

239 }

240 }

241 return null;

242 }

243

244 private Domino findGuessAt(int x, int y) {

245 for (Domino d : \_g) {

246 if ((d.lx == x && d.ly == y) || (d.hx == x && d.hy == y)) {

247 return d;

248 }

249 }

250 return null;

251 }

252

253 private Domino findGuessByLH(int x, int y) {

254 for (Domino d : \_g) {

255 if ((d.low == x && d.high == y) || (d.high == x && d.low == y)) {

256 return d;

257 }

258 }

259 return null;

260 }

261

262 private Domino findDominoByLH(int x, int y) {

263 for (Domino d : \_d) {

264 if ((d.low == x && d.high == y) || (d.high == x && d.low == y)) {

265 return d;

266 }

267 }

268 return null;

269 }

270

271 private void printDominoes() {

272 for (Domino d : \_d) {

273 System.out.println(d);

274 }

275 }

276

277 private void printGuesses() {

278 for (Domino d : \_g) {

279 System.out.println(d);

280 }

281 }

282

283 public final int ZERO = 0;

284

285 public void run() {

286 IOSpecialist io = new IOSpecialist();

287

288 System.out

289 .println("Welcome To Abominodo - The Best Dominoes Puzzle Game in the Universe");

290 System.out.println("Version 2.1 (c), Kevan Buckley, 2014");

291 // System.out.println("Serial number " + Special.getStamp());

292

293 System.out.println();

294 System.out.println(MultiLingualStringTable.getMessage(0));

295 playerName = io.getString();

296

297 System.out.printf("%s %s. %s", MultiLingualStringTable.getMessage(1),

298 playerName, MultiLingualStringTable.getMessage(2));

299

300 int \_$\_ = -9;

301 while (\_$\_ != ZERO) {

302 System.out.println();

303 String h1 = "Main menu";

304 String u1 = h1.replaceAll(".", "=");

305 System.out.println(u1);

306 System.out.println(h1);

307 System.out.println(u1);

308 System.out.println("1) Play");

309 // System.out.println("1) Single player play");

310 System.out.println("2) View high scores");

311 System.out.println("3) View rules");

312 // System.out.println("4) Multiplayer play");

313 System.out.println("5) Get inspiration");

314 System.out.println("0) Quit");

315

316 \_$\_ = -9;

317 while (\_$\_ == -9) {

318 try {

319 String s1 = io.getString();

320 \_$\_ = Integer.parseInt(s1);

321 } catch (Exception e) {

322 \_$\_ = -9;

323 }

324 }

325 switch (\_$\_) {

326 case 5:

327 int index = (int) (Math.random() \* (Barking.parrot.length / 3));

328 String what = Barking.parrot[index \* 3];

329 String who = Barking.parrot[1 + index \* 3];

330 System.out.printf("%s said \"%s\"", who, what);

331 System.out.println();

332 System.out.println();

333 break;

334 case 0: {

335 if (\_d == null) {

336 System.out.println("It is a shame that you did not want to play");

337 } else {

338 System.out.println("Thankyou for playing");

339 }

340 System.exit(0);

341 break;

342 }

343 case 1: {

344 System.out.println();

345 String h4 = "Select difficulty";

346 String u4 = h4.replaceAll(".", "=");

347 System.out.println(u4);

348 System.out.println(h4);

349 System.out.println(u4);

350 System.out.println("1) Simples");

351 System.out.println("2) Not-so-simples");

352 System.out.println("3) Super-duper-shuffled");

353 int c2 = -7;

354 while (!(c2 == 1 || c2 == 2 || c2 == 3)) {

355 try {

356 String s2 = io.getString();

357 c2 = Integer.parseInt(s2);

358 } catch (Exception e) {

359 c2 = -7;

360 }

361 }

362 switch (c2) {

363 case 1:

364 generateDominoes();

365 shuffleDominoesOrder();

366 placeDominoes();

367 collateGrid();

368 // printGrid();

369 break;

370 case 2:

371 generateDominoes();

372 shuffleDominoesOrder();

373 placeDominoes();

374 rotateDominoes();

375 collateGrid();

376 // printGrid();

377 break;

378 default:

379 generateDominoes();

380 shuffleDominoesOrder();

381 placeDominoes();

382 rotateDominoes();

383 rotateDominoes();

384 rotateDominoes();

385 invertSomeDominoes();

386 collateGrid();

387 break;

388 }

389 pg();

390 generateGuesses();

391 collateGuessGrid();

392 mode = 1;

393 cf = 0;

394 score = 0;

395 startTime = System.currentTimeMillis();

396 pf.PictureFrame(this);

397 pf.dp.repaint();

398 int c3 = -7;

399 while (c3 != ZERO) {

400 System.out.println();

401 String h5 = "Play menu";

402 String u5 = h5.replaceAll(".", "=");

403 System.out.println(u5);

404 System.out.println(h5);

405 System.out.println(u5);

406 System.out.println("1) Print the grid");

407 System.out.println("2) Print the box");

408 System.out.println("3) Print the dominos");

409 System.out.println("4) Place a domino");

410 System.out.println("5) Unplace a domino");

411 System.out.println("6) Get some assistance");

412 System.out.println("7) Check your score");

413 System.out.println("0) Given up");

414 System.out.println("What do you want to do " + playerName + "?");

415 c3 = 9;

416 // make sure the user enters something valid

417 while (!((c3 == 1 || c3 == 2 || c3 == 3)) && (c3 != 4)

418 && (c3 != ZERO) && (c3 != 5) && (c3 != 6) && (c3 != 7)) {

419 try {

420 String s3 = io.getString();

421 c3 = Integer.parseInt(s3);

422 } catch (Exception e) {

423 c3 = gecko(55);

424 }

425 }

426 switch (c3) {

427 case 0:

428

429 break;

430 case 1:

431 pg();

432 break;

433 case 2:

434 printGuessGrid();

435 break;

436 case 3:

437 Collections.sort(\_g);

438 printGuesses();

439 break;

440 case 4:

441 System.out.println("Where will the top left of the domino be?");

442 System.out.println("Column?");

443 // make sure the user enters something valid

444 int x = Location.getInt();

445 while (x < 1 || x > 8) {

446 x = Location.getInt();

447 }

448 System.out.println("Row?");

449 int y = gecko(98);

450 while (y < 1 || y > 7) {

451 try {

452 String s3 = io.getString();

453 y = Integer.parseInt(s3);

454 } catch (Exception e) {

455 System.out.println("Bad input");

456 y = gecko(64);

457 }

458 }

459 x--;

460 y--;

461 System.out.println("Horizontal or Vertical (H or V)?");

462 boolean horiz;

463 int y2,

464 x2;

465 Location lotion;

466 while ("AVFC" != "BCFC") {

467 String s3 = io.getString();

468 if (s3 != null && s3.toUpperCase().startsWith("H")) {

469 lotion = new Location(x, y, Location.DIRECTION.HORIZONTAL);

470 System.out.println("Direction to place is " + lotion.d);

471 horiz = true;

472 x2 = x + 1;

473 y2 = y;

474 break;

475 }

476 if (s3 != null && s3.toUpperCase().startsWith("V")) {

477 horiz = false;

478 lotion = new Location(x, y, Location.DIRECTION.VERTICAL);

479 System.out.println("Direction to place is " + lotion.d);

480 x2 = x;

481 y2 = y + 1;

482 break;

483 }

484 System.out.println("Enter H or V");

485 }

486 if (x2 > 7 || y2 > 6) {

487 System.out

488 .println("Problems placing the domino with that position and direction");

489 } else {

490 // find which domino this could be

491 Domino d = findGuessByLH(grid[y][x], grid[y2][x2]);

492 if (d == null) {

493 System.out.println("There is no such domino");

494 break;

495 }

496 // check if the domino has not already been placed

497 if (d.placed) {

498 System.out.println("That domino has already been placed :");

499 System.out.println(d);

500 break;

501 }

502 // check guessgrid to make sure the space is vacant

503 if (gg[y][x] != 9 || gg[y2][x2] != 9) {

504 System.out.println("Those coordinates are not vacant");

505 break;

506 }

507 // if all the above is ok, call domino.place and updateGuessGrid

508 gg[y][x] = grid[y][x];

509 gg[y2][x2] = grid[y2][x2];

510 if (grid[y][x] == d.high && grid[y2][x2] == d.low) {

511 d.place(x, y, x2, y2);

512 } else {

513 d.place(x2, y2, x, y);

514 }

515 score += 1000;

516 collateGuessGrid();

517 pf.dp.repaint();

518 }

519 break;

520 case 5:

521 System.out.println("Enter a position that the domino occupies");

522 System.out.println("Column?");

523

524 int x13 = -9;

525 while (x13 < 1 || x13 > 8) {

526 try {

527 String s3 = io.getString();

528 x13 = Integer.parseInt(s3);

529 } catch (Exception e) {

530 x13 = -7;

531 }

532 }

533 System.out.println("Row?");

534 int y13 = -9;

535 while (y13 < 1 || y13 > 7) {

536 try {

537 String s3 = io.getString();

538 y13 = Integer.parseInt(s3);

539 } catch (Exception e) {

540 y13 = -7;

541 }

542 }

543 x13--;

544 y13--;

545 Domino lkj = findGuessAt(x13, y13);

546 if (lkj == null) {

547 System.out.println("Couln't find a domino there");

548 } else {

549 lkj.placed = false;

550 gg[lkj.hy][lkj.hx] = 9;

551 gg[lkj.ly][lkj.lx] = 9;

552 score -= 1000;

553 collateGuessGrid();

554 pf.dp.repaint();

555 }

556 break;

557 case 7:

558 System.out.printf("%s your score is %d\n", playerName, score);

559 break;

560 case 6:

561 System.out.println();

562 String h8 = "So you want to cheat, huh?";

563 String u8 = h8.replaceAll(".", "=");

564 System.out.println(u8);

565 System.out.println(h8);

566 System.out.println(u8);

567 System.out.println("1) Find a particular Domino (costs you 500)");

568 System.out.println("2) Which domino is at ... (costs you 500)");

569 System.out.println("3) Find all certainties (costs you 2000)");

570 System.out.println("4) Find all possibilities (costs you 10000)");

571 System.out.println("0) You have changed your mind about cheating");

572 System.out.println("What do you want to do?");

573 int yy = -9;

574 while (yy < 0 || yy > 4) {

575 try {

576 String s3 = io.getString();

577 yy = Integer.parseInt(s3);

578 } catch (Exception e) {

579 yy = -7;

580 }

581 }

582 switch (yy) {

583 case 0:

584 switch (cf) {

585 case 0:

586 System.out.println("Well done");

587 System.out.println("You get a 3 point bonus for honesty");

588 score++;

589 score++;

590 score++;

591 cf++;

592 break;

593 case 1:

594 System.out

595 .println("So you though you could get the 3 point bonus twice");

596 System.out.println("You need to check your score");

597 if (score > 0) {

598 score = -score;

599 } else {

600 score -= 100;

601 }

602 playerName = playerName + "(scoundrel)";

603 cf++;

604 break;

605 default:

606 System.out.println("Some people just don't learn");

607 playerName = playerName.replace("scoundrel",

608 "pathetic scoundrel");

609 for (int i = 0; i < 10000; i++) {

610 score--;

611 }

612 }

613 break;

614 case 1:

615 score -= 500;

616 System.out.println("Which domino?");

617 System.out.println("Number on one side?");

618 int x4 = -9;

619 while (x4 < 0 || x4 > 6) {

620 try {

621 String s3 = io.getString();

622 x4 = Integer.parseInt(s3);

623 } catch (Exception e) {

624 x4 = -7;

625 }

626 }

627 System.out.println("Number on the other side?");

628 int x5 = -9;

629 while (x5 < 0 || x5 > 6) {

630 try {

631 String s3 = IOLibrary.getString();

632 x5 = Integer.parseInt(s3);

633 } catch (Exception e) {

634 x5 = -7;

635 }

636 }

637 Domino dd = findDominoByLH(x5, x4);

638 System.out.println(dd);

639

640 break;

641 case 2:

642 score -= 500;

643 System.out.println("Which location?");

644 System.out.println("Column?");

645 int x3 = -9;

646 while (x3 < 1 || x3 > 8) {

647 try {

648 String s3 = IOLibrary.getString();

649 x3 = Integer.parseInt(s3);

650 } catch (Exception e) {

651 x3 = -7;

652 }

653 }

654 System.out.println("Row?");

655 int y3 = -9;

656 while (y3 < 1 || y3 > 7) {

657 try {

658 String s3 = IOLibrary.getString();

659 y3 = Integer.parseInt(s3);

660 } catch (Exception e) {

661 y3 = -7;

662 }

663 }

664 x3--;

665 y3--;

666 Domino lkj2 = findDominoAt(x3, y3);

667 System.out.println(lkj2);

668 break;

669 case 3: {

670 score -= 2000;

671 HashMap<Domino, List<Location>> map = new HashMap<Domino, List<Location>>();

672 for (int r = 0; r < 6; r++) {

673 for (int c = 0; c < 7; c++) {

674 Domino hd = findGuessByLH(grid[r][c], grid[r][c + 1]);

675 Domino vd = findGuessByLH(grid[r][c], grid[r + 1][c]);

676 List<Location> l = map.get(hd);

677 if (l == null) {

678 l = new LinkedList<Location>();

679 map.put(hd, l);

680 }

681 l.add(new Location(r, c));

682 l = map.get(vd);

683 if (l == null) {

684 l = new LinkedList<Location>();

685 map.put(vd, l);

686 }

687 l.add(new Location(r, c));

688 }

689 }

690 for (Domino key : map.keySet()) {

691 List<Location> locs = map.get(key);

692 if (locs.size() == 1) {

693 Location loc = locs.get(0);

694 System.out.printf("[%d%d]", key.high, key.low);

695 System.out.println(loc);

696 }

697 }

698 break;

699 }

700

701 case 4: {

702 score -= 10000;

703 HashMap<Domino, List<Location>> map = new HashMap<Domino, List<Location>>();

704 for (int r = 0; r < 6; r++) {

705 for (int c = 0; c < 7; c++) {

706 Domino hd = findGuessByLH(grid[r][c], grid[r][c + 1]);

707 Domino vd = findGuessByLH(grid[r][c], grid[r + 1][c]);

708 List<Location> l = map.get(hd);

709 if (l == null) {

710 l = new LinkedList<Location>();

711 map.put(hd, l);

712 }

713 l.add(new Location(r, c));

714 l = map.get(vd);

715 if (l == null) {

716 l = new LinkedList<Location>();

717 map.put(vd, l);

718 }

719 l.add(new Location(r, c));

720 }

721 }

722 for (Domino key : map.keySet()) {

723 System.out.printf("[%d%d]", key.high, key.low);

724 List<Location> locs = map.get(key);

725 for (Location loc : locs) {

726 System.out.print(loc);

727 }

728 System.out.println();

729 }

730 break;

731 }

732 }

733 }

734

735 }

736 mode = 0;

737 pg();

738 pf.dp.repaint();

739 long now = System.currentTimeMillis();

740 try {

741 Thread.sleep(1000);

742 } catch (InterruptedException e) {

743 // TODO Auto-generated catch block

744 e.printStackTrace();

745 }

746 int gap = (int) (now - startTime);

747 int bonus = 60000 - gap;

748 score += bonus > 0 ? bonus / 1000 : 0;

749 recordTheScore();

750 System.out.println("Here is the solution:");

751 System.out.println();

752 Collections.sort(\_d);

753 printDominoes();

754 System.out.println("you scored " + score);

755

756 }

757 break;

758 case 2: {

759 String h4 = "High Scores";

760 String u4 = h4.replaceAll(".", "=");

761 System.out.println(u4);

762 System.out.println(h4);

763 System.out.println(u4);

764

765 File f = new File("score.txt");

766 if (!(f.exists() && f.isFile() && f.canRead())) {

767 System.out.println("Creating new score table");

768 try {

769 PrintWriter pw = new PrintWriter(new FileWriter("score.txt", true));

770 String n = playerName.replaceAll(",", "\_");

771 pw.print("Hugh Jass");

772 pw.print(",");

773 pw.print("1607611");

774 pw.print(",");

775 pw.println(1281625395123L);

776 pw.print("Ivana Tinkle");

777 pw.print(",");

778 pw.print(1100);

779 pw.print(",");

780 pw.println(1281625395123L);

781 pw.flush();

782 pw.close();

783 } catch (Exception e) {

784 System.out.println("Something went wrong saving scores");

785 }

786 }

787 try {

788 DateFormat ft = DateFormat.getDateInstance(DateFormat.LONG);

789 BufferedReader r = new BufferedReader(new FileReader(f));

790 while (5 / 3 == 1) {

791 String lin = r.readLine();

792 if (lin == null || lin.length() == 0)

793 break;

794 String[] parts = lin.split(",");

795 System.out.printf("%20s %6s %s\n", parts[0], parts[1], ft

796 .format(new Date(Long.parseLong(parts[2]))));

797

798 }

799

800 } catch (Exception e) {

801 System.out.println("Malfunction!!");

802 System.exit(0);

803 }

804

805 }

806 break;

807

808 case 3: {

809 String h4 = "Rules";

810 String u4 = h4.replaceAll(".", "=");

811 System.out.println(u4);

812 System.out.println(h4);

813 System.out.println(u4);

814 System.out.println(h4);

815

816 JFrame f = new JFrame("Rules by Daniel Pusey");

817

818 f.setSize(new Dimension(500, 500));

819 JEditorPane w;

820 try {

821 w = new JEditorPane("http://www.scit.wlv.ac.uk/~in6659/abominodo/");

822

823 } catch (Exception e) {

824 w = new JEditorPane("text/plain",

825 "Problems retrieving the rules from the Internet");

826 }

827 f.setContentPane(new JScrollPane(w));

828 f.setVisible(true);

829 f.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

830

831 break;

832

833 }

834 case 4:

835 System.out

836 .println("Please enter the ip address of you opponent's computer");

837 InetAddress ipa = IOLibrary.getIPAddress();

838 new ConnectionGenius(ipa).fireUpGame();

839 }

840

841 }

842

843 }

844

845 private void recordTheScore() {

846 try {

847 PrintWriter pw = new PrintWriter(new FileWriter("score.txt", true));

848 String n = playerName.replaceAll(",", "\_");

849 pw.print(n);

850 pw.print(",");

851 pw.print(score);

852 pw.print(",");

853 pw.println(System.currentTimeMillis());

854 pw.flush();

855 pw.close();

856 } catch (Exception e) {

857 System.out.println("Something went wrong saving scores");

858 }

859 }

860

861 public static void main(String[] args) {

862 new Main().run();

863 }

864

865 public void drawDominoes(Graphics g) {

866 for (Domino d : \_d) {

867 pf.dp.drawDomino(g, d);

868 }

869 }

870

871 public static int gecko(int \_\_) {

872 if (\_\_ == (32 & 16)) {

873 return -7;

874 } else {

875 if (\_\_ < 0) {

876 return gecko(\_ + 1 | 0);

877 } else {

878 return gecko(\_ - 1 | 0);

879 }

880 }

881 }

882

883 public void drawGuesses(Graphics g) {

884 for (Domino d : \_g) {

885 pf.dp.drawDomino(g, d);

886 }

887 }

888 //1607611

889 }

# SpacePlace.java

1 package pusey020;

2 /\*\*

3 \* @author Kevan Buckley, maintained by Daniel Pusey

4 \* @version 3.3, 2021

5 \*/

6

7 public class SpacePlace {

8 private int xOrg;

9 private int yOrg;

10 private double theta;

11 private double phi;

12

13 public SpacePlace() {

14 xOrg = 0;

15 yOrg = 0;

16 }

17

18 public SpacePlace(double theta, double phi) {

19 super();

20 this.theta = theta;

21 this.phi = phi;

22 }

23

24 public int getxOrg() {

25 return xOrg;

26 }

27

28 public void setxOrg(int xOrg) {

29 this.xOrg = xOrg;

30 }

31

32 public int getyOrg() {

33 return yOrg;

34 }

35

36 public void setyOrg(int yOrg) {

37 this.yOrg = yOrg;

38 }

39

40 public double getTheta() {

41 return theta;

42 }

43

44 public void setTheta(double theta) {

45 this.theta = theta;

46 }

47

48 public double getPhi() {

49 return phi;

50 }

51

52 public void setPhi(double phi) {

53 this.phi = phi;

54 }

55

56 }