```java
package catan;

import java.awt.Polygon;
/**
 * This class holds methods and variables associated with the hexes used to create the board in Catan.
 * Contains a built in GUI, unusable without the user interface.
 *
 * @author Eddie Gurnee
 * @author Nicole Downer
 * @version 0.0.25 10/26/2013
 * @see CatanFrame
 * @see HexBoard
 *
 */
public class Hex extends JPanel {
    /**
     * This thrice-be-damned class holds all of the method and variables needed to cause me to go insane.
     *
     * @author Eddie Gurnee
     * @version 0.0.18 10/28/2013
     * @see Road
     */
    private class Edge extends BoardSpace {
        private final int B_WIDTH = 30;
        private final int B_HEIGHT = 15;

        private final int S_WIDTH = 10;
        private final int S_HEIGHT = 30;

        private int edgeNum;

        private Edge(int edgeNum) {
            super();

            this.setAllNotEligible();
            this.edgeNum = edgeNum;

            if (edgeNum == 0) {
                //mid right
```

```
45                  setSize(S_WIDTH, S_HEIGHT);
46              } else {
47                  //bot right and left
48                  setSize(B_WIDTH, B_HEIGHT);
49              }
50          }
51          @Override
52          public void actionPerformed(ActionEvent e) {
53              int n = JOptionPane.showConfirmDialog
54                      (null,
55                              "Would you like to place a road?",
56                              "Place road",
57                              JOptionPane.YES_NO_OPTION
58                              );
59
60              if (n == 0) {
61                  try {
62                      if (!
   BoardGame.getActivePlayer().isRemainingRoads()) {
63                          throw new NoMorePiecesException("Roads");
64                      }
65                      if (!
   BoardGame.getActivePlayer().isRoadBuilding()) {
66                          if (!
   BoardGame.getActivePlayer().isEnoughMoney(new
   Road(BoardGame.getActivePlayer(), -1).getCost())) {
67                              throw new
   NotEnoughResourcesException("Road");
68                          }
69                      }
70                      if (!isEligible()) {
71                          throw new NoConnectionException("Road");
72                      }
73                      this.onLanding();
74                  } catch (NoMorePiecesException|
   NoConnectionException|NotEnoughResourcesException ex) {
75                      ex.displayErrMessage();
76                  }
77              }
78          }
79          @Override
```

```
80          public void onLanding() {
81              Road theRoad = new Road(BoardGame.getActivePlayer(),
    this.edgeNum);
82
83              if (BoardGame.start) {
84                  if (!BoardGame.getActivePlayer().isTheRoadPlaced())
    {
85
    BoardGame.getActivePlayer().setTheRoadPlaced(true);
86                  } else if (!
    BoardGame.getActivePlayer().isSecondRoadPlaced()) {
87
    BoardGame.getActivePlayer().setSecondRoadPlaced(true);
88                  }
89              }
90
91              //enable nearby edges
92              int r = Hex.this.getRow();
93              int c = Hex.this.getCol();
94
95              switch (edgeNum) {
96              case 0:
97                  Hex.this.enableEdge(1);
98                  Hex.this.enableCorner(0);
99                  try {
100                     BoardGame.board.gameBoard[r][c +
    1].enableEdge(2);
101                 } catch (ArrayIndexOutOfBoundsException ex) {
102                     //Don't you worry child
103                 }
104                 try {
105                     if (r <= 3) {
106                         BoardGame.board.gameBoard[r - 1]
    [c].enableCorner(1);
107                         BoardGame.board.gameBoard[r - 1]
    [c].enableEdge(1);
108                     } else {
109                         BoardGame.board.gameBoard[r - 1][c +
    1].enableCorner(1);
110                         BoardGame.board.gameBoard[r - 1][c +
    1].enableEdge(1);
```

```
111                     }
112                 } catch (ArrayIndexOutOfBoundsException ex) {
113                     //Don't you worry child
114                 }
115                 try {
116                     if (r <= 3) {
117                         BoardGame.board.gameBoard[r - 1]
    [c].enableEdge(2);
118                     } else {
119                         BoardGame.board.gameBoard[r - 1][c +
    1].enableEdge(2);
120                     }
121                 } catch (ArrayIndexOutOfBoundsException ex) {
122                     //Don't you worry child
123                 }
124                 break;
125             case 1:
126                 Hex.this.enableEdge(0);
127                 Hex.this.enableEdge(2);
128                 Hex.this.enableCorner(0, 1);
129                 try {
130                     BoardGame.board.gameBoard[r][c +
    1].enableEdge(2);
131                 } catch (ArrayIndexOutOfBoundsException ex) {
132                     //Don't you worry child
133                 }
134                 try{
135                     if (r < 3) {
136                         BoardGame.board.gameBoard[r + 1]
    [c].enableEdge(0);
137                     } else {
138                         BoardGame.board.gameBoard[r + 1][c -
    1].enableEdge(0);
139                     }
140                 } catch (ArrayIndexOutOfBoundsException ex) {
141                     //Don't you worry child
142                 }
143                 break;
144             case 2:
145                 Hex.this.enableEdge(1);
146                 Hex.this.enableCorner(1);
```

```
147                 try {
148                     BoardGame.board.gameBoard[r][c -
    1].enableCorner(1);
149                     BoardGame.board.gameBoard[r][c -
    1].enableEdge(0, 1);
150                 } catch (ArrayIndexOutOfBoundsException ex) {
151                     //Don't you worry child
152                 }
153                 try {
154                     if (r < 3) {
155                         BoardGame.board.gameBoard[r + 1]
    [c].enableEdge(0);
156                     } else {
157                         BoardGame.board.gameBoard[r + 1][c -
    1].enableEdge(0);
158                     }
159                 } catch (ArrayIndexOutOfBoundsException ex) {
160                     //Don't you worry child
161                 }
162                 break;
163             }
164 
165             Hex.this.add(theRoad);
166             theRoad.setBounds(this.getBounds());
167 
168             if (BoardGame.getActivePlayer().isRoadBuilding()) {
169                 BoardGame.getActivePlayer().builtFreeRoad();
170                 BoardGame.getActivePlayer().placedRoad();
171             } else {
172                 BoardGame.getActivePlayer().buyPiece(theRoad);
173             }
174 
175             Hex.this.remove(this);
176         }
177     }
178     /**
179      * This class holds the methods and variable associated with the
    corners of each hex.
180      * Cities and Settlements go here.
181      *
182      * @author Eddie Gurnee
```

```
183      * @version 0.0.41 10/27/2013
184      * @see Properties
185      *
186      */
187     private class Corner extends BoardSpace {
188         private final int WIDTH = 20;
189         private final int HEIGHT = 20;
190         private int corNum;
191 
192         private Corner() {
193             this(-1);
194         }
195         private Corner(int corNum) {
196             super();
197             setSize(WIDTH, HEIGHT);
198 
199             this.corNum = corNum;
200         }
201         @Override
202         public void actionPerformed(ActionEvent e) {
203             int n = JOptionPane.showConfirmDialog
204                     (null,
205                             "Would you like to place a settlement?",
206                             "Place Settlement",
207                             JOptionPane.YES_NO_OPTION
208                             );
209             if (n == 0) {
210                 try {
211                     if (!
    BoardGame.getActivePlayer().isEnoughMoney(new
    Properties(BoardGame.getActivePlayer()).getCost())) {
212                         throw new
    NotEnoughResourcesException("Settlement");
213                     }
214                     if (!this.isEligible()) {
215                         throw new
    NoConnectionException("Settlement");
216                     }
217                     if (!
    BoardGame.getActivePlayer().isRemainingSettlements()) {
218                         throw new
```

```
   NoMorePiecesException("Settlements");
219                     }
220                     this.onLanding();
221                 } catch (NoMorePiecesException|
   NoConnectionException|NotEnoughResourcesException ex) {
222                     ex.displayErrMessage();
223                 }
224             }
225         }
226         @Override
227         public void onLanding() {
228 //          System.out.println("Hex : " + Hex.this.getRow() + " " +
   Hex.this.getCol()
229 //                  + "\nCorner Num: " + corNum + " CLICKED!");
230 
231             Properties theSettlement = new
   Properties(BoardGame.getActivePlayer());
232 
233             Hex.this.addProperty(theSettlement);
234 
235             int r = Hex.this.getRow();
236             int c = Hex.this.getCol();
237 
238             if (corNum == 0) {
239                 //remove adjacent corners
240                 Hex.this.removeCorner(1);
241                 try {
242                     BoardGame.board.gameBoard[r][c +
   1].removeCorner(1);
243                 } catch (ArrayIndexOutOfBoundsException ex) {
244                     //Don't you worry child
245                 }
246                 try {
247                     if (r > 3) {
248                         BoardGame.board.gameBoard[r - 1][c +
   1].removeCorner(1);
249                     } else {
250                         BoardGame.board.gameBoard[r - 1]
   [c].removeCorner(1);
251                     }
252                 } catch (ArrayIndexOutOfBoundsException ex) {
```

```
253                     //Don't you worry child
254                 }
255 
256                 //enable nearby edges
257                 Hex.this.enableEdge(0);
258                 Hex.this.enableEdge(1);
259                 BoardGame.board.gameBoard[r][c + 1].enableEdge(2);
260 
261                 //add properties to the adjacent Hexes
262                 BoardGame.board.gameBoard[r][c +
    1].addProperty(theSettlement);
263                 if (r < 3) {
264                     BoardGame.board.gameBoard[r + 1][c +
    1].addProperty(theSettlement);
265                 } else {
266                     BoardGame.board.gameBoard[r + 1]
    [c].addProperty(theSettlement);
267                 }
268             } else {
269                 //remove adjacent corners
270                 Hex.this.removeCorner(0);
271                 try {
272                     BoardGame.board.gameBoard[r][c -
    1].removeCorner(0);
273                 } catch (ArrayIndexOutOfBoundsException ex) {
274                     //Don't you worry child
275                 }
276                 try {
277                     if (r >= 3) {
278                         BoardGame.board.gameBoard[r + 1][c -
    1].removeCorner(0);
279                     } else {
280                         BoardGame.board.gameBoard[r + 1]
    [c].removeCorner(0);
281                     }
282                 } catch (ArrayIndexOutOfBoundsException ex) {
283                     //Don't you worry child
284                 }
285 
286                 //enable nearby edges
287                 Hex.this.enableEdge(1);
```

```
288                 Hex.this.enableEdge(2);
289                 try {
290                     if (r >= 3) {
291                         BoardGame.board.gameBoard[r + 1][c -
    1].enableEdge(0);
292                     } else {
293                         BoardGame.board.gameBoard[r + 1]
    [c].enableEdge(0);
294                     }
295                 } catch (ArrayIndexOutOfBoundsException ex) {
296                     //Don't you worry child
297                 }
298 
299                 //add properties to nearby hexes
300                 if (r < 3) {
301                     BoardGame.board.gameBoard[r + 1]
    [c].addProperty(theSettlement);
302                     BoardGame.board.gameBoard[r + 1][c +
    1].addProperty(theSettlement);
303                 } else {
304                     BoardGame.board.gameBoard[r + 1]
    [c].addProperty(theSettlement);
305                     BoardGame.board.gameBoard[r + 1][c -
    1].addProperty(theSettlement);
306                 }
307             }
308 
309             Hex.this.add(theSettlement);
310             theSettlement.setBounds(this.getBounds());
311 
312             BoardGame.getActivePlayer().buyPiece(theSettlement);
313 
314             Hex.this.remove(this);
315             BoardGame.board.repaint();
316         }
317     }
318 
319     private final int WIDTH = 100, HEIGHT = 100, OFFSET = 10;
320     private final int SIDES = 6;
321 
322     private final int NUM_EDGES = SIDES / 2;
```

```
323     private final int NUM_CORNERS = SIDES / 3;
324 
325     private final int[] XPOINTS = {
326             (WIDTH / 2) + OFFSET,
327             (WIDTH) + OFFSET,
328             (WIDTH) + OFFSET,
329             (WIDTH / 2) + OFFSET,
330             (0) + OFFSET,
331             (0) + OFFSET,
332     };
333     private final int[] YPOINTS = {
334             (0) + OFFSET,
335             (HEIGHT / 4) + OFFSET,
336             ((3 * HEIGHT) / 4) + OFFSET,
337             (HEIGHT) + OFFSET,
338             ((3 * HEIGHT) / 4) + OFFSET,
339             (HEIGHT / 4) + OFFSET,
340     };
341     private final Polygon HEX = new Polygon (XPOINTS, YPOINTS,
    SIDES);
342 
343     private String type;
344 
345     private int row;
346     private int col;
347 
348     private Corner[] corners = new Corner[NUM_CORNERS];
349     private Edge[] edges = new Edge[NUM_EDGES];
350 
351     private ArrayList<Properties> establishedProperties = new
    ArrayList<Properties>();
352 
353     public Hex(String type) {
354         super();
355         setSize((WIDTH) + (2 * OFFSET), (HEIGHT) + (2 *OFFSET));
356         setOpaque(false);
357         setLayout(null);
358 
359         this.type = type;
360 
361         for (int i = 0; i < NUM_CORNERS; i++) {
```

```
362             corners[i] = new Corner(i);
363             this.add(corners[i]);
364         }
365         for (int i = 0; i < NUM_EDGES; i++) {
366             edges[i] = new Edge(i);
367             this.add(edges[i]);
368         }
369 
370         placeCorners();
371         placeEdges();
372     }
373     private void placeCorners() {
374         for (int i = 0; i < NUM_CORNERS; i++) {
375             if (i == 0) {
376                 corners[i].setBounds(
377                         (this.getWidth() - corners[i].getWidth()),
378                         ((3 * (this.getHeight() -
    corners[i].getHeight())) / 4),
379                         corners[i].getWidth(),
380                         corners[i].getHeight());
381             } else if (i == 1) {
382                 corners[i].setBounds(
383                         ((this.getWidth() - corners[i].getWidth()) /
    2),
384                         (this.getHeight() - corners[i].getHeight()),
385                         corners[i].getWidth(),
386                         corners[i].getHeight());
387             }
388         }
389     }
390     private void placeEdges() {
391         for (int i = 0; i < NUM_EDGES; i++) {
392             switch (i) {
393             //          default:
394             case 0:
395                 edges[i].setBounds(
396                         (OFFSET + WIDTH - (edges[i].getWidth() /
    2)),
397                         (OFFSET + (HEIGHT / 4) + (new
    Corner().HEIGHT / 2)),
398                         edges[i].getWidth(),
```

```
399                         edges[i].getHeight());
400                 break;
401             case 1:
402                 edges[i].setBounds(
403                         (OFFSET + (WIDTH / 2) + (new
    Corner().WIDTH / 2)),
404                         (OFFSET + (3 * HEIGHT / 4) + (new
    Corner().HEIGHT / 4)),
405                         edges[i].getWidth(),
406                         edges[i].getHeight());
407                 break;
408             case 2:
409                 edges[i].setBounds(
410                         (OFFSET + (new Corner().WIDTH / 2)),
411                         (OFFSET + (3 * HEIGHT / 4) + (new
    Corner().HEIGHT / 4)),
412                         edges[i].getWidth(),
413                         edges[i].getHeight());
414                 break;
415             }
416         }
417     }
418     public void enableEdge(int... edgeNum) {
419         for (int edge : edgeNum) {
420             this.edges[edge].setEligible(true);
421         }
422     }
423     public void removeEdge(int... edgeNum) {
424         for (int edge : edgeNum) {
425             this.remove(edges[edge]);
426         }
427     }
428     public void removeCorner(int... cornerNum) {
429         for (int corner : cornerNum) {
430             this.remove(corners[corner]);
431         }
432     }
433     public void enableCorner(int... cornerNum) {
434         for (int corner : cornerNum) {
435             this.corners[corner].setEligible(true);
436         }
```

```
437     }
438     public void enableCorners() {
439         for (int c = 0; c < corners.length; c++) {
440             corners[c].setAllEligible();
441         }
442     }
443     public void disableCorners() {
444         for (Corner c: corners) {
445             c.setAllNotEligible();
446         }
447     }
448     public String getType() {
449         return type;
450     }
451     public Polygon getHex() {
452         return HEX;
453     }
454     protected void setType(String type) {
455         this.type = type;
456     }
457     protected ArrayList<Properties> getProperties() {
458         return establishedProperties;
459     }
460     public void addProperty(Properties newProperty) {
461         establishedProperties.add(newProperty);
462     }
463     public void setRowAndCol(int row, int col) {
464         this.row = row;
465         this.col = col;
466     }
467     public int getRow() {
468         return row;
469     }
470     public int getCol() {
471         return col;
472     }
473     public String toString() {
474         return "Type: " + this.type + "\nRow: " + this.row + "\nCol:
    " + this.col;
475     }
476 }
```