```
    1: package dev.gavinthomas.tictactoe.opponents;
    2:
    3: import java.awt.Point;
    4: import java.util.List;
    5: import java.util.ArrayList;
    6: import java.util.function.Consumer;
    7:
    8: import dev.gavinthomas.tictactoe.types.Opponent;
    9: import dev.gavinthomas.tictactoe.Board;
   10: import dev.gavinthomas.tictactoe.Board.PieceType;
   11: import dev.gavinthomas.tictactoe.utils.Minimax;
   12: import out.Out;
   13:
   14: public class Computer implements Opponent {
   15:   private int moves;
   16:   public final PieceType PIECE;
   17:   public final Minimax AI;
   18:   private final Board board;
   19:   private final Consumer<Point> callback;
   20:   public Computer(Board board, final PieceType PIECE, Consumer<Point> callback)
{
   21:     this.callback = callback;
   22:     this.board = board;
   23:     this.PIECE = PIECE;
   24:     this.AI = new Minimax(PIECE);
   25:   }
   26:
   27:   public PieceType getPiece() {
   28:     return PIECE;
   29:   }
   30:
   31:   public void getMove() {
   32: //    if (board.tempGetCompMove == false) return null;
   33: //    try {
   34: //      Thread.sleep((long) (Math.random() * 1500) + 1000);
   35: //    } catch (InterruptedException ignore) {}
   36:
   37:     callback.accept(AI.getBest(board.grid));
   38:     // return new Point();
   39:   }
   40:
   41:   public boolean canMove(int x, int y, MoveOption moveOp) {
   42:     try {
   43:       for (Point pt : moveOp.openLocs) {
   44:         if (board.grid[x + pt.x][y + pt.y] != PieceType.BLANK) {
   45:           return false;
   46:         }
   47:       }
   48:       for (Point pt : moveOp.ownLocs) {
   49:         if (board.grid[x + pt.x][y + pt.y] != PIECE) {
   50:           return false;
   51:         }
   52:       }
   53:
   54:       System.out.println(board.grid[0][0]);
   55:       for (Point pt : moveOp.oppLocs) {
   56:         Out.append(board.grid[x + pt.x][y + pt.y]);
   57:         if (board.grid[x + pt.x][y + pt.y] != (PIECE == PieceType.X ? PieceType.
O : PieceType.X)) {
   58:           return false;
   59:         }
   60:       }
   61:     } catch (IndexOutOfBoundsException e) {
```

```
62:         // System.out.println("IOBE");
63:         return false;
64:       }
65:       return true;
66:     }
67:
68:
69:
70:     public static class MoveOption {
71:       public int priority;
72:       public List<Point> openLocs = new ArrayList<Point>();
73:       public List<Point> ownLocs = new ArrayList<Point>();
74:       public List<Point> oppLocs = new ArrayList<Point>();
75:
76:       public MoveOption(int priority) {
77:         this.priority = priority;
78:       }
79:
80:       public MoveOption open(int x, int y) {
81:         this.openLocs.add(new Point(x, y));
82:         return this;
83:       }
84:
85:       public MoveOption own(int x, int y) {
86:         this.ownLocs.add(new Point(x, y));
87:         return this;
88:       }
89:
90:       public MoveOption opp(int x, int y) {
91:         this.oppLocs.add(new Point(x, y));
92:         return this;
93:       }
94:     }
95: }
96:
97:
```

```
    1: package dev.gavinthomas.tictactoe.opponents;
    2:
    3: import java.awt.Point;
    4: import java.util.function.Consumer;
    5:
    6: import dev.gavinthomas.tictactoe.types.Opponent;
    7: import dev.gavinthomas.tictactoe.input.Keycode;
    8: import dev.gavinthomas.tictactoe.Board;
    9: import dev.gavinthomas.tictactoe.Board.PieceType;
   10:
   11: public class Player implements Opponent {
   12:   private final Board board;
   13:   private final Point selectedSpot = new Point(0, 0);
   14:   private volatile boolean canMove = false;
   15:   private final Consumer<Point> callback;
   16:   public final PieceType PIECE;
   17:
   18:   public Player(Board board, PieceType PIECE, Consumer<Point> callback) {
   19:     this.callback = callback;
   20:     this.board = board;
   21:     this.PIECE = PIECE;
   22:   }
   23:
   24:   public PieceType getPiece() {
   25:     return PIECE;
   26:   }
   27:
   28:   public void handleInput(Object[] args) {
   29:     if (!canMove) return;
   30:     Keycode key = (Keycode) args[0];
   31:     if (key == Keycode.UP_ARROW && selectedSpot.y != 2) {
   32:       board.highlightSpot(selectedSpot.x, selectedSpot.y, false);
   33:       board.highlightSpot(selectedSpot.x, selectedSpot.y + 1, true);
   34:       selectedSpot.translate(0, 1);
   35:     } else if (key == Keycode.DOWN_ARROW && selectedSpot.y != 0) {
   36:       board.highlightSpot(selectedSpot.x, selectedSpot.y, false);
   37:       board.highlightSpot(selectedSpot.x, selectedSpot.y - 1, true);
   38:       selectedSpot.translate(0, -1);
   39:     } else if (key == Keycode.LEFT_ARROW && selectedSpot.x != 0) {
   40:       board.highlightSpot(selectedSpot.x, selectedSpot.y, false);
   41:       board.highlightSpot(selectedSpot.x - 1, selectedSpot.y, true);
   42:       selectedSpot.translate(-1, 0);
   43:     } else if (key == Keycode.RIGHT_ARROW && selectedSpot.x != 2) {
   44:       board.highlightSpot(selectedSpot.x, selectedSpot.y, false);
   45:       board.highlightSpot(selectedSpot.x + 1, selectedSpot.y, true);
   46:       selectedSpot.translate(1, 0);
   47:     } else if (key == Keycode.SPACE && board.grid[selectedSpot.x][selectedSpot.y
] == PieceType.BLANK) {
   48:     // } else if (key == Keycode.SPACE) {
   49:       // System.out.println(selectedSpot.x + ", " + selectedSpot.y);
   50:       this.canMove = false;
   51:       board.highlightSpot(selectedSpot.x, selectedSpot.y, false);
   52:       this.callback.accept(selectedSpot);
   53:     }
   54:   }
   55:
   56:   public void getMove() {
   57:     board.highlightSpot(selectedSpot.x, selectedSpot.y, true);
   58:     this.canMove = true;
   59:   }
   60:
   61:   public void getMove(int x, int y) {
   62:     selectedSpot.x = x;
```

```
    63:     selectedSpot.y = y;
    64:     getMove();
    65:   }
    66: }
```

```java
package dev.gavinthomas.tictactoe.types;

import dev.gavinthomas.tictactoe.Board.PieceType;

public interface Opponent {
  public void getMove();

  public PieceType getPiece();
}
```

```
    1: package dev.gavinthomas.tictactoe.types;
    2:
    3: import dev.gavinthomas.tictactoe.ui.SelectionUI;
    4:
    5: import java.text.MessageFormat;
    6:
    7: public abstract class Visuals {
    8: //  private static final int[] xBlockVals2 = { 255, 0, 0 };
    9:   private static final String[] gridVals = { "\033[1B\033[58D" };
   10:   private static final String[] xBlockVals = { "\033[38;2;255;0;0m", "\033[0m\03
3[1B\033[10D" };
   11:   private static final String[] oBlockVals = { "\033[38;2;255;255;255m", "\033[0
m\033[1B\033[10D" };
   12:   private static final String[] blankBlockVals = { "\033[0m", "\033[1B\033[10D"
};
   13:   private static final String[] highlightVals = { "\033[38;2;0;255;0;1m", "\033[
1B\033[18D", "\033[16C" };
   14:   private static final String[] titleVals = {"", "\033[1B\033[70D", "\033[31m",
"\033[0m"};
   15:   private static final String[] sizeUIVals = { "", "\033[1B\033[19D" };
   16:   private static final String[] TITLELTRS = {
   17:       """
   18: â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\225
\227
   19: â\225\232â\225\220â\225\220â\226\210â\226\210â\225\224â\225\220â\225\220â\225
\235
   20: â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226\221â\226
\221
   21: â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226\221â\226
\221
   22: â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226\221â\226
\221
   23: â\226\221â\226\221â\226\221â\225\232â\225\220â\225\235â\226\221â\226\221â\226
\221
   24: """, """
   25: â\226\210â\226\210â\225\227
   26: â\226\210â\226\210â\225\221
   27: â\226\210â\226\210â\225\221
   28: â\226\210â\226\210â\225\221
   29: â\226\210â\226\210â\225\221
   30: â\225\232â\225\220â\225\235
   31: """, """
   32: â\226\221â\226\210â\226\210â\226\210â\226\210â\226\210â\225\227â\226\221
   33: â\226\210â\226\210â\225\224â\225\220â\225\220â\226\210â\226\210â\225\227
   34: â\226\210â\226\210â\225\221â\226\221â\226\221â\225\232â\225\220â\225\235
   35: â\226\210â\226\210â\225\221â\226\221â\226\221â\226\210â\226\210â\225\227
   36: â\225\232â\226\210â\226\210â\226\210â\226\210â\226\210â\225\224â\225\235
   37: â\226\221â\225\232â\225\220â\225\220â\225\220â\225\220â\225\235â\226\221
   38: """, """
   39: â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\225
\227
   40: â\225\232â\225\220â\225\220â\226\210â\226\210â\225\224â\225\220â\225\220â\225
\235
   41: â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226\221â\226
\221
   42: â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226\221â\226
\221
   43: â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226\221â\226
\221
   44: â\226\221â\226\221â\226\221â\225\232â\225\220â\225\235â\226\221â\226\221â\226
\221
   45: """, """
   46: â\226\221â\226\210â\226\210â\226\210â\226\210â\226\210â\225\227â\226\221
```

```
   47: â\226\210â\226\210â\225\224â\225\220â\225\220â\226\210â\226\210â\225\227
   48: â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\225\221
   49: â\226\210â\226\210â\225\224â\225\220â\225\220â\226\210â\226\210â\225\221
   50: â\226\210â\226\210â\225\221â\226\221â\226\221â\226\210â\226\210â\225\221
   51: â\225\232â\225\220â\225\235â\226\221â\226\221â\225\232â\225\220â\225\235
   52: """, """
   53: â\226\221â\226\210â\226\210â\226\210â\226\210â\226\210â\225\227â\226\221
   54: â\226\210â\226\210â\225\224â\225\220â\225\220â\226\210â\226\210â\225\227
   55: â\226\210â\226\210â\225\221â\226\221â\226\221â\225\232â\225\220â\225\235
   56: â\226\210â\226\210â\225\221â\226\221â\226\221â\226\210â\226\210â\225\227
   57: â\225\232â\226\210â\226\210â\226\210â\226\210â\226\210â\225\224â\225\235
   58: â\226\221â\225\232â\225\220â\225\220â\225\220â\225\220â\225\235â\226\221
   59: """, """
   60: â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\225
\227
   61: â\225\232â\225\220â\225\220â\226\210â\226\210â\225\224â\225\220â\225\220â\225
\235
   62: â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226\221â\226
\221
   63: â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226\221â\226
\221
   64: â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226\221â\226
\221
   65: â\226\221â\226\221â\226\221â\225\232â\225\220â\225\235â\226\221â\226\221â\226
\221
   66: """, """
   67: â\226\221â\226\210â\226\210â\226\210â\226\210â\226\210â\225\227â\226\221
   68: â\226\210â\226\210â\225\224â\225\220â\225\220â\226\210â\226\210â\225\227
   69: â\226\210â\226\210â\225\221â\226\221â\226\221â\226\210â\226\210â\225\221
   70: â\226\210â\226\210â\225\221â\226\221â\226\221â\226\210â\226\210â\225\221
   71: â\225\232â\226\210â\226\210â\226\210â\226\210â\226\210â\225\224â\225\235
   72: â\226\221â\225\232â\225\220â\225\220â\225\220â\225\220â\225\235â\226\221
   73: """, """
   74: â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\225\227
   75: â\226\210â\226\210â\225\224â\225\220â\225\220â\225\220â\225\220â\225\235
   76: â\226\210â\226\210â\226\210â\226\210â\226\210â\225\227â\226\221â\226\221
   77: â\226\210â\226\210â\225\224â\225\220â\225\220â\225\235â\226\221â\226\221
   78: â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\225\227
   79: â\225\232â\225\220â\225\220â\225\220â\225\220â\225\220â\225\220â\225\235
   80: """};
   81:
   82:   public static final String GRID = MessageFormat.format("" +
   83:       "                   â\226\210â\226\210                   â\226\210â\226\210
           {0}" +
   84:       "                   â\226\210â\226\210                   â\226\210â\226\210
           {0}" +
   85:       "                   â\226\210â\226\210                   â\226\210â\226\210
           {0}" +
   86:       "                   â\226\210â\226\210                   â\226\210â\226\210
           {0}" +
   87:       "                   â\226\210â\226\210                   â\226\210â\226\210
           {0}" +
   88:       "                   â\226\210â\226\210                   â\226\210â\226\210
           {0}" +
   89:       "                   â\226\210â\226\210                   â\226\210â\226\210
           {0}" +
   90:       "                   â\226\210â\226\210                   â\226\210â\226\210
           {0}" +
   91:       "                   â\226\210â\226\210                   â\226\210â\226\210
           {0}" +
   92:       "â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â
\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226
\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â
```

```
\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226
\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â
\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226
\210â\226\210â\226\210{0}" +
    93:         "                    â\226\210â\226\210                    â\226\210â\226\210
                    {0}" +
    94:         "                    â\226\210â\226\210                    â\226\210â\226\210
                    {0}" +
    95:         "                    â\226\210â\226\210                    â\226\210â\226\210
                    {0}" +
    96:         "                    â\226\210â\226\210                    â\226\210â\226\210
                    {0}" +
    97:         "                    â\226\210â\226\210                    â\226\210â\226\210
                    {0}" +
    98:         "                    â\226\210â\226\210                    â\226\210â\226\210
                    {0}" +
    99:         "                    â\226\210â\226\210                    â\226\210â\226\210
                    {0}" +
   100:         "                    â\226\210â\226\210                    â\226\210â\226\210
                    {0}" +
   101:         "                    â\226\210â\226\210                    â\226\210â\226\210
                    {0}" +
   102:         "â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â
\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226
\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â
\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226
\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â
\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226
\210â\226\210â\226\210{0}" +
   103:         "                    â\226\210â\226\210                    â\226\210â\226\210
                    {0}" +
   104:         "                    â\226\210â\226\210                    â\226\210â\226\210
                    {0}" +
   105:         "                    â\226\210â\226\210                    â\226\210â\226\210
                    {0}" +
   106:         "                    â\226\210â\226\210                    â\226\210â\226\210
                    {0}" +
   107:         "                    â\226\210â\226\210                    â\226\210â\226\210
                    {0}" +
   108:         "                    â\226\210â\226\210                    â\226\210â\226\210
                    {0}" +
   109:         "                    â\226\210â\226\210                    â\226\210â\226\210
                    {0}" +
   110:         "                    â\226\210â\226\210                    â\226\210â\226\210
                    {0}" +
   111:         "                    â\226\210â\226\210                    â\226\210â\226\210
                ",
   112:         (Object[]) gridVals);
   113:
   114:     public static final String XBLOCK = MessageFormat.format("" +
   115:         "{0} â\226\204â\226\204â\226\204  â\226\204â\226\204â\226\204 {1}" +
   116:         "{0}  â\226\210â\226\210â\226\204â\226\204â\226\210â\226\210  {1}" +
   117:         "{0}   â\226\210â\226\210â\226\210â\226\210   {1}" +
   118:         "{0}    â\226\210â\226\210    {1}" +
   119:         "{0}   â\226\210â\226\210â\226\210â\226\210   {1}" +
   120:         "{0}  â\226\210â\226\210  â\226\210â\226\210  {1}" +
   121:         "{0} â\226\200â\226\200â\226\200  â\226\200â\226\200â\226\200 {1}",
   122:         (Object[]) xBlockVals);
   123:
   124:     public static final String OBLOCK = MessageFormat.format("" +
   125:         "{0}   â\226\204â\226\204â\226\204â\226\204   {1}" +
   126:         "{0}  â\226\210â\226\210â\226\200â\226\200â\226\210â\226\210  {1}" +
   127:         "{0} â\226\210â\226\210    â\226\210â\226\210 {1}" +
```

```
  128:        "{0} â\226\210â\226\210    â\226\210â\226\210 {1}" +
  129:        "{0} â\226\210â\226\210    â\226\210â\226\210 {1}" +
  130:        "{0}  â\226\210â\226\210â\226\204â\226\204â\226\210â\226\210  {1}" +
  131:        "{0}   â\226\200â\226\200â\226\200â\226\200   {1}",
  132:        (Object[]) oBlockVals);
  133:
  134:    public static final String BLANKBLOCK = MessageFormat.format("" +
  135:        "{0}          {1}" +
  136:        "{0}          {1}" +
  137:        "{0}          {1}" +
  138:        "{0}          {1}" +
  139:        "{0}          {1}" +
  140:        "{0}          {1}" +
  141:        "{0}          {1}",
  142:        (Object[]) blankBlockVals);
  143:
  144:    public static final String HIGHLIGHT = MessageFormat.format("" +
  145:        "{0}â\225-â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200
\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\225®{
1}" +
  146:        "{0}â\224\202{2}â\224\202{1}" +
  147:        "{0}â\224\202{2}â\224\202{1}" +
  148:        "{0}â\224\202{2}â\224\202{1}" +
  149:        "{0}â\224\202{2}â\224\202{1}" +
  150:        "{0}â\224\202{2}â\224\202{1}" +
  151:        "{0}â\224\202{2}â\224\202{1}" +
  152:        "{0}â\224\202{2}â\224\202{1}" +
  153:
  154:        "{0}â\225°â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â
\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\225¯"
,
  155:        (Object[]) highlightVals);
  156:
  157:    public static final String NOHIGHLIGHT = MessageFormat.format("" +
  158:            "{0}â\225-â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224
\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â
\225®{1}" +
  159:            "{0}â\224\202{2}â\224\202{1}" +
  160:            "{0}â\224\202{2}â\224\202{1}" +
  161:            "{0}â\224\202{2}â\224\202{1}" +
  162:            "{0}â\224\202{2}â\224\202{1}" +
  163:            "{0}â\224\202{2}â\224\202{1}" +
  164:            "{0}â\224\202{2}â\224\202{1}" +
  165:            "{0}â\224\202{2}â\224\202{1}" +
  166:
  167:            "{0}â\225°â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224
\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â
\225¯",
  168:        (Object[]) highlightVals);
  169:
  170:    public static final String UNHIGHLIGHT = MessageFormat.format("" +
  171:        "{0}                  {1}" +
  172:        "{0} {2} {1}" +
  173:        "{0} {2} {1}" +
  174:        "{0} {2} {1}" +
  175:        "{0} {2} {1}" +
  176:        "{0} {2} {1}" +
  177:        "{0} {2} {1}" +
  178:        "{0} {2} {1}" +
  179:
  180:        "{0}                  ",
  181:        (Object[]) highlightVals);
  182:
```

```
  183:
  184:   public static final String TITLE = MessageFormat.format("" +
  185:       "{0}â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226
\210â\225\227â\226\210â\226\210â\225\227â\226\221â\226\210â\226\210â\226\210â\226\210â
\226\210â\225\227â\226\221" + "â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â
\226\210â\226\210â\225\227â\226\221â\226\210â\226\210â\226\210â\226\210â\226\210â\225
\227â\226\221â\226\221â\226\210â\226\210â\226\210â\226\210â\226\210â\225\227â\226\221".
replaceAll("â\226\210", "\033[31mâ\226\210\033[0m") + "â\226\210â\226\210â\226\210â\226
\210â\226\210â\226\210â\226\210â\226\210â\225\227â\226\221â\226\210â\226\210â\226\210â
\226\210â\226\210â\225\227â\226\221â\226\210â\226\210â\226\210â\226\210â\226\210â\226
\210â\226\210â\225\227{1}" +
  186:       "{0}â\225\232â\225\220â\225\220â\226\210â\226\210â\225\224â\225\220â\225
\220â\225\235â\226\210â\226\210â\225\221â\226\210â\226\210â\225\224â\225\220â\225\220â
\226\210â\226\210â\225\227" + "â\225\232â\225\220â\225\220â\226\210â\226\210â\225\224â
\225\220â\225\220â\225\235â\226\210â\226\210â\225\224â\225\220â\225\220â\226\210â\226
\210â\225\227â\226\210â\226\210â\225\224â\225\220â\225\220â\226\210â\226\210â\225\227".
replaceAll("â\226\210", "\033[31mâ\226\210\033[0m") + "â\225\232â\225\220â\225\220â\226
\210â\226\210â\225\224â\225\220â\225\220â\225\235â\226\210â\226\210â\225\224â\225\220â
\225\220â\226\210â\226\210â\225\227â\226\210â\226\210â\225\224â\225\220â\225\220â\225
\220â\225\220â\225\235{1}" +
  187:       "{0}â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226
\221â\226\221â\226\210â\226\210â\225\221â\226\210â\226\210â\225\221â\226\221â\226\221â
\225\232â\225\220â\225\235" + "â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â
\226\221â\226\221â\226\221â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226
\210â\225\221â\226\210â\226\210â\225\221â\226\221â\226\221â\225\232â\225\220â\225\235".
replaceAll("â\226\210", "\033[31mâ\226\210\033[0m") + "â\226\221â\226\221â\226\221â\226
\210â\226\210â\225\221â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â
\226\221â\226\210â\226\210â\225\221â\226\210â\226\210â\226\210â\226\210â\226\210â\225
\227â\226\221â\226\221{1}" +
  188:       "{0}â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226
\221â\226\221â\226\210â\226\210â\225\221â\226\210â\226\210â\225\221â\226\221â\226\221â
\226\210â\226\210â\225\227" + "â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â
\226\221â\226\221â\226\221â\226\210â\226\210â\225\224â\225\220â\225\220â\226\210â\226
\210â\225\221â\226\210â\226\210â\225\221â\226\221â\226\221â\226\210â\226\210â\225\227".
replaceAll("â\226\210", "\033[31mâ\226\210\033[0m") + "â\226\221â\226\221â\226\221â\226
\210â\226\210â\225\221â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â
\226\221â\226\210â\226\210â\225\221â\226\210â\226\210â\225\224â\225\220â\225\220â\225
\235â\226\221â\226\221{1}" +
  189:       "{0}â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226
\221â\226\221â\226\210â\226\210â\225\221â\225\232â\226\210â\226\210â\226\210â\226\210â
\226\210â\225\224â\225\235" + "â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â
\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226\221â\226\210â\226
\210â\225\221â\225\232â\226\210â\226\210â\226\210â\226\210â\226\210â\225\224â\225\235".
replaceAll("â\226\210", "\033[31mâ\226\210\033[0m") + "â\226\221â\226\221â\226\221â\226
\210â\226\210â\225\221â\226\221â\226\221â\226\221â\225\232â\226\210â\226\210â\226\210â
\226\210â\226\210â\225\224â\225\235â\226\210â\226\210â\226\210â\226\210â\226\210â\226
\210â\226\210â\225\227{1}" +
  190:       "{0}â\226\221â\226\221â\226\221â\225\232â\225\220â\225\235â\226\221â\226
\221â\226\221â\225\232â\225\220â\225\235â\226\221â\225\232â\225\220â\225\220â\225\220â
\225\220â\225\235â\226\221" + "â\226\221â\226\221â\226\221â\225\232â\225\220â\225\235â
\226\221â\226\221â\226\221â\225\232â\225\220â\225\235â\226\221â\226\221â\225\232â\225
\220â\225\235â\226\221â\225\232â\225\220â\225\220â\225\220â\225\220â\225\235â\226\221".
replaceAll("â\226\210", "\033[31mâ\226\210\033[0m") + "â\226\221â\226\221â\226\221â\225
\232â\225\220â\225\235â\226\221â\226\221â\226\221â\226\221â\225\232â\225\220â\225\220â
\225\220â\225\220â\225\235â\226\221â\225\232â\225\220â\225\220â\225\220â\225\220â\225
\220â\225\220â\225\235{1}",
  191:       (Object[]) titleVals);
  192:
  193:
  194:
  195:   public static String title1(boolean[] reds) {
  196:     String[] vals = { "\033[0m\033[1B\033[20D",
  197:         (reds[0] ? "\033[38;2;255;0;0m" : "\033[37m"),
```

```
  198:          (reds[1] ? "\033[38;2;255;0;0m" : "\033[37m"),
  199:          (reds[2] ? "\033[38;2;255;0;0m" : "\033[37m")
  200:      };
  201:
  202:      return MessageFormat.format("" +
  203:          "{1}â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226
\210â\225\227{2}â\226\210â\226\210â\225\227{3}â\226\221â\226\210â\226\210â\226\210â\226
\210â\226\210â\225\227â\226\221{0}" +
  204:          "{1}â\225\232â\225\220â\225\220â\226\210â\226\210â\225\224â\225\220â\225
\220â\225\235{2}â\226\210â\226\210â\225\221{3}â\226\210â\226\210â\225\224â\225\220â\225
\220â\226\210â\226\210â\225\227{0}" +
  205:          "{1}â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226
\221â\226\221{2}â\226\210â\226\210â\225\221{3}â\226\210â\226\210â\225\221â\226\221â\226
\221â\225\232â\225\220â\225\235{0}" +
  206:          "{1}â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226
\221â\226\221{2}â\226\210â\226\210â\225\221{3}â\226\210â\226\210â\225\221â\226\221â\226
\221â\226\210â\226\210â\225\227{0}" +
  207:          "{1}â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226
\221â\226\221{2}â\226\210â\226\210â\225\221{3}â\225\232â\226\210â\226\210â\226\210â\226
\210â\226\210â\225\224â\225\235{0}" +
  208:          "{1}â\226\221â\226\221â\226\221â\225\232â\225\220â\225\235â\226\221â\226
\221â\226\221{2}â\225\232â\225\220â\225\235{3}â\226\221â\225\232â\225\220â\225\220â\225
\220â\225\220â\225\235â\226\221\033[0m",
  209:          (Object[]) vals).replaceAll("â\226\221", "\033[37mâ\226\221\033[0m");
  210:    }
  211:
  212:    public static String title2(String color) {
  213:      String[] vals = { "\033[" + color + "m", "\033[1B\033[25D" };
  214:
  215:      return MessageFormat.format("" +
  216:              "{0}â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â
\226\210â\225\227â\226\221â\226\210â\226\210â\226\210â\226\210â\226\210â\225\227â\226
\221â\226\221â\226\210â\226\210â\226\210â\226\210â\226\210â\225\227â\226\221{1}" +
  217:              "{0}â\225\232â\225\220â\225\220â\226\210â\226\210â\225\224â\225\220â
\225\220â\225\235â\226\210â\226\210â\225\224â\225\220â\225\220â\226\210â\226\210â\225
\227â\226\210â\226\210â\225\224â\225\220â\225\220â\226\210â\226\210â\225\227{1}" +
  218:              "{0}â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â
\226\221â\226\221â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\225
\221â\226\210â\226\210â\225\221â\226\221â\226\221â\225\232â\225\220â\225\235{1}" +
  219:              "{0}â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â
\226\221â\226\221â\226\210â\226\210â\225\224â\225\220â\225\220â\226\210â\226\210â\225
\221â\226\210â\226\210â\225\221â\226\221â\226\221â\226\210â\226\210â\225\227{1}" +
  220:              "{0}â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â
\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226\221â\226\210â\226\210â\225
\221â\225\232â\226\210â\226\210â\226\210â\226\210â\226\210â\225\224â\225\235{1}" +
  221:              "{0}â\226\221â\226\221â\226\221â\225\232â\225\220â\225\235â\226\221â
\226\221â\226\221â\225\232â\225\220â\225\235â\226\221â\226\221â\225\232â\225\220â\225
\235â\226\221â\225\232â\225\220â\225\220â\225\220â\225\220â\225\235â\226\221",
  222:          (Object[]) vals);
  223:    }
  224:
  225:    public static String title3(String color) {
  226:      String[] vals = { "\033[" + color + "m", "\033[1B\033[25D" };
  227:
  228:      return MessageFormat.format("" +
  229:              "{0}â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â
\226\210â\225\227â\226\221â\226\210â\226\210â\226\210â\226\210â\226\210â\225\227â\226
\221â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\225\227{1}" +
  230:              "{0}â\225\232â\225\220â\225\220â\226\210â\226\210â\225\224â\225\220â
\225\220â\225\235â\226\210â\226\210â\225\224â\225\220â\225\220â\226\210â\226\210â\225
\227â\226\210â\226\210â\225\224â\225\220â\225\220â\225\220â\225\220â\225\235{1}" +
  231:              "{0}â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â
\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226\221â\226\210â\226\210â\225
```

```
\221â\226\210â\226\210â\226\210â\226\210â\226\210â\225\227â\226\221â\226\221{1}" +
  232:             "{0}â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â
\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â\226\221â\226\210â\226\210â\225
\221â\226\210â\226\210â\225\224â\225\220â\225\220â\225\235â\226\221â\226\221{1}" +
  233:             "{0}â\226\221â\226\221â\226\221â\226\210â\226\210â\225\221â\226\221â
\226\221â\226\221â\225\232â\226\210â\226\210â\226\210â\226\210â\226\210â\225\224â\225
\235â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\226\210â\225\227{1}" +
  234:             "{0}â\226\221â\226\221â\226\221â\225\232â\225\220â\225\235â\226\221â
\226\221â\226\221â\226\221â\225\232â\225\220â\225\220â\225\220â\225\220â\225\235â\226
\221â\225\232â\225\220â\225\220â\225\220â\225\220â\225\220â\225\220â\225\235",
  235:         (Object[]) vals);
  236:   }
  237:
  238:   public static String title(boolean[] reds) {
  239:     StringBuilder finStr = new StringBuilder();
  240:     for (int i = 0; i < TITLELTRS.length; i++) {
  241:       String[] vals = { "\033[1B\033[" + (TITLELTRS[i].indexOf('\n')) + "D",
  242:           (reds[i] ? "\033[38;2;255;0;0m" : "\033[37m"),
  243:           "\033[38;2;128;128;128m", "\033[0m", "\033[1C\033[5A"
  244:       };
  245: //      System.out.println(vals[1].replaceAll("\033", ""));
  246:       StringBuilder newLtrBldr = new StringBuilder();
  247:       for (int j = 0; j != -1; j = TITLELTRS[i].indexOf('\n', j + 1)) {
  248:         if (TITLELTRS[i].indexOf('\n', j + 1) == -1) break;
  249:         newLtrBldr.append(TITLELTRS[i], j + (j == 0 ? 0 : 1), TITLELTRS[i].index
Of('\n', j + 1)).append("{3}");
  250:         int nextLine = TITLELTRS[i].indexOf('\n', j + 1);
  251:         if (TITLELTRS[i].indexOf('\n', nextLine + 1) != -1) newLtrBldr.append("{
0}");
  252:       }
  253:       newLtrBldr.append("{4}");
  254: //      String newStr = newLtrBldr.toString();
  255: //      String newStr = newLtrBldr.toString().replaceAll("(â\226\221)+", "{2}$0{
3}");
  256:       String newStr = newLtrBldr.toString().replaceAll("â\226\221", " ");
  257:       newStr = newStr.replaceAll("([â\225\224â\225\220â\225\235â\225\221â\225
\232â\225\227â\226\210])+", "{1}$0{3}");
  258: //      System.out.println(newStr);
  259: //      System.out.println(MessageFormat.format(newStr, (Object[]) vals));
  260:       finStr.append(MessageFormat.format(newStr, (Object[]) vals));
  261:     }
  262:     return finStr.toString();
  263:   }
  264:
  265:   public static String sizeUI(int x, int y, int rx, int ry) {
  266:     String xStr = " ".repeat(7 - String.valueOf(x).length()) + x;
  267:     String yStr = y + " ".repeat(7 - String.valueOf(y).length());
  268:     String rxStr = " ".repeat(7 - String.valueOf(rx).length()) + rx;
  269:     String ryStr = ry + " ".repeat(7 - String.valueOf(ry).length());
  270:
  271:     return MessageFormat.format("" +
  272:         "{0}â\225-â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224
\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â
\224\200â\225®{1}" +
  273:         "{0}â\224\202     \033[1;4mMinimum\033[0m     â\224\202{1}" +
  274:         "{0}â\224\202" + rxStr + " x " + ryStr + "â\224\202{1}" +
  275:         "{0}â\224\202     \033[1;4mCurrent\033[0m     â\224\202{1}" +
  276:         "{0}â\224\202" + xStr + " x " + yStr + "â\224\202{1}" +
  277:         "{0}â\225°â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224
\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â\224\200â
\224\200â\225¯{1}",
  278:
  279:         (Object[]) sizeUIVals);
```

```
  280:   }
  281:
  282:   public static String menuButton(String text, SelectionUI.MODE mode) {
  283:     String[] vals = { "\033[0m\033[1B\033[20D",
  284:         (mode == SelectionUI.MODE.DISABLED ? "\033[2m" : ""),
  285:         (mode == SelectionUI.MODE.SELECTED ? "\033[1m" : ""),
  286:         "\033[0m"
  287:     };
  288:     text = " ".repeat((text.length() % 2 == 0 ? 9 : 8) - (text.length() / 2)) +
  289:         text + " ".repeat(9 - (text.length() / 2));
  290:     String[] lines = (mode == SelectionUI.MODE.SELECTED ?
  291:         new String[]{"â\224\201", "â\224\203", "â\224\217", "â\224\223", "â\224
\233", "â\224\227"} :
  292:         new String[]{"â\224\200", "â\224\202", "â\224\214", "â\224\220", "â\224
\230", "â\224\224"});
  293:
  294:     return MessageFormat.format("" +
  295:         "{1}" + lines[2] + lines[0].repeat(18) + lines[3] + "{0}" +
  296:         "{1}" + lines[1] + "{2}" + text + "{3}" + lines[1] + "{0}" +
  297:         "{1}" + lines[5] + lines[0].repeat(18) + lines[4] + "\033[0m",
  298:         (Object[]) vals);
  299:   }
  300:
  301:
  302:   public static String box(int w, int h) {
  303:     String[] vals = {"\033[1B\033[" + w + "D", "\033[" + (w - 4) + "C"};
  304:     return MessageFormat.format("" +
  305:         "â\226\210".repeat(w) + "{0}" +
  306:         "â\226\210â\226\210{1}â\226\210â\226\210{0}".repeat(h - 2) +
  307:         "â\226\210".repeat(w),
  308:         (Object[]) vals);
  309:   }
  310:
  311:   public static String doubleLineBox(int w, int h) {
  312:     String[] vals = {"\033[1B\033[" + w + "D", "\033[" + (w - 2) + "C"};
  313:     return MessageFormat.format("" +
  314:             "â\225\224" + "â\225\220".repeat(w - 2) + "â\225\227" + "{0}" +
  315:             "â\225\221{1}â\225\221{0}".repeat(h - 2) +
  316:             "â\225\232" + "â\225\220".repeat(w - 2) + "â\225\235",
  317:         (Object[]) vals);
  318:   }
  319:
  320: }
```

```java
package dev.gavinthomas.tictactoe.types;

import java.awt.Point;

public interface UIHolder {

  Point offset();
  Point size();
  void render();

//  public Point getOffset();
}
```

```
 1: package dev.gavinthomas.tictactoe.types;
 2:
 3: import java.awt.Point;
 4:
 5: public interface UIComponent {
 6:   void render();
 7:   void endTasks();
 8: }
 9:
10:
```

```java
package dev.gavinthomas.tictactoe.utils;

import java.util.List;
import java.util.ArrayList;

import dev.gavinthomas.tictactoe.opponents.Computer.MoveOption;

public abstract class MoveList {
  public static List<MoveOption> list = new ArrayList<MoveOption>();

  static {
    // opponent win positions
    {
      // vertical
      {
        list.add(new MoveOption(9).opp(0, -1).opp(0, 1));
        list.add(new MoveOption(9).opp(0, 1).opp(0, 2));
        list.add(new MoveOption(9).opp(0, -1).opp(0, -2));
      }

      // horizontal
      {
        list.add(new MoveOption(9).opp(-1, 0).opp(1, 0));
        list.add(new MoveOption(9).opp(1, 0).opp(2, 0));
        list.add(new MoveOption(9).opp(-1, 0).opp(-2, 0));
      }

      // diagonal
      {
        list.add(new MoveOption(9).opp(-1, -1).opp(1, 1));
        list.add(new MoveOption(9).opp(1, 1).opp(2, 2));
        list.add(new MoveOption(9).opp(-1, -1).opp(-2, -2));
        list.add(new MoveOption(9).opp(1, -1).opp(2, -2));
        list.add(new MoveOption(9).opp(1, -1).opp(-1, 1));
        list.add(new MoveOption(9).opp(-1, 1).opp(-2, 2));
      }
    }
  }
}
```

```java
package dev.gavinthomas.tictactoe.utils;

import dev.gavinthomas.tictactoe.input.InputQueue;
import dev.gavinthomas.tictactoe.input.input;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.HashMap;
import java.util.function.Consumer;

public class Term {
  private final Map<String, int[]> cursorSaves = new HashMap<String, int[]>();
  private final List<String> queue = new ArrayList<>();
  public boolean enabled = true;

  public void printQueue() {
    for (String s : queue) {
      System.out.print(s);
    }
  }

  public void saveCursor() {
    this.print("\033[s");
  }

  public void restoreCursor() {
    this.print("\033[u");
  }

  public void setCursorPos(int x, int y) {
    this.print("\033[" + y + ";" + x + "H");
  }

  public void clear(boolean goHome) {
    this.print("\033[2J" + (goHome ? "\033[H" : ""));
  }

  public void hideCursor(boolean tog) {
    this.print("\033[?25" + (tog ? "l" : "h"));
  }

  public void print(String outp) {
    if (!enabled) {
      queue.add(outp);
      return;
    }
    System.out.print(outp);
  }

  public void requestSize(Consumer<Object[]> callback) {
    this.saveCursor();
    this.setCursorPos(50000, 50000);
    this.print("\033[6n");
    this.restoreCursor();
    input.queue.add(new InputQueue.TermSize.Builder(callback).args(new Object[]{
InputQueue.ArgType.TERMSIZE}).build());
  }
}
```

```
    1: package dev.gavinthomas.tictactoe.utils;
    2:
    3: import java.lang.reflect.Method;
    4: import java.lang.reflect.Proxy;
    5: import java.lang.reflect.InvocationHandler;
    6:
    7: public abstract class listeners {
    8:   public static class terminalResizeListener {
    9:     final Runnable runner;
   10:
   11:     private static class Handler implements InvocationHandler {
   12:       @Override
   13:       public Object invoke(Object proxy, Method method, Object[] args) {
   14:         return null;
   15:       }
   16:     }
   17:
   18:     private void register() throws Throwable {
   19:
   20:     }
   21:
   22:     public terminalResizeListener(final Runnable runner) {
   23:       this.runner = runner;
   24:       try {
   25:         Class<?> signalClass = Class.forName("sun.misc.Signal");
   26:         for (Method m : signalClass.getDeclaredMethods()) {
   27:           if ("handle".equals(m.getName())) {
   28:             Object windowResizeHandler = Proxy.newProxyInstance(getClass().getCl
assLoader(),
   29:                 new Class[] { Class.forName("sun.misc.SignalHandler") }, (proxy,
 method, args) -> {
   30:                   if ("handle".equals(method.getName())) {
   31:                     runner.run();
   32:                   }
   33:                   return null;
   34:                 });
   35:             m.invoke(null, signalClass.getConstructor(String.class).newInstance(
"WINCH"), windowResizeHandler);
   36:           }
   37:         }
   38:       } catch (Throwable ignore) {}
   39:     }
   40:   }
   41:
   42:   public static class onExitListener {
   43:     public onExitListener(Runnable runner) {
   44:       Runtime.getRuntime().addShutdownHook(new Thread(runner));
   45:     }
   46:   }
   47: }
```

```java
package dev.gavinthomas.tictactoe.utils;

import dev.gavinthomas.tictactoe.Board.PieceType;
import dev.gavinthomas.tictactoe.TTT;

import java.awt.Point;
import java.util.Arrays;

public class Minimax {
  private final PieceType ownPiece, oppPiece;

  public Minimax(PieceType ownPiece) {
    this.ownPiece = ownPiece;
    this.oppPiece = (ownPiece == PieceType.X ? PieceType.O : PieceType.X);
  }

  public int eval(PieceType[][] board, int turns) {
    PieceType winner = TTT.getWinner(board);
    if (winner == ownPiece) {
      return 10 - turns;
    } else if (winner == oppPiece) {
      return -10 - turns;
    } else {
      return 0;
    }
  }

  public int minimax(PieceType[][] board, int turns, boolean ownTurn, int alpha,
int beta) {
    int evalVal = eval(board, turns);
    if (evalVal == 0 && TTT.gameOver(board)) {
      return 0;
    } else if (evalVal != 0) {
      return evalVal;
    }

    int best = (ownTurn ? -1000 : 1000);

    if (turns >= 10) return best;
    for (int x = 0; x < board.length; x++) {
      for (int y = 0; y < board[x].length; y++) {
        if (board[x][y] != PieceType.BLANK) continue;
//        System.out.println(x + ", " + y);
        board[x][y] = (ownTurn ? ownPiece : oppPiece);
        int retVal = minimax(board, turns + 1, !ownTurn, alpha, beta);
        best = (ownTurn ? Math.max(best, retVal) : Math.min(best, retVal));
        board[x][y] = PieceType.BLANK;
        if (ownTurn) {
          alpha = Math.max(alpha, best);
        } else {
          beta = Math.min(beta, best);
        }
        if (alpha >= beta) return best;
      }
    }
    return best;

  }

  public Point getBest(PieceType[][] boardArr) {
    int best = -1000;
    Point bestMove = null;

```

```
63:     PieceType[][] board = Arrays.copyOf(boardArr, boardArr.length);
64:
65:     for (int x = 0; x < board.length; x++) {
66:       for (int y = 0; y < board[x].length; y++) {
67:         if (board[x][y] != PieceType.BLANK) continue;
68:         board[x][y] = ownPiece;
69:         int move = minimax(board, 0, false, -1000, 1000);
70:         board[x][y] = PieceType.BLANK;
71: //        System.out.println(move + " > " + best);
72: //        System.out.print(x + ", " + y + ": " + move + " | ");
73:         if (move >= best) {
74:           bestMove = new Point(x, y);
75:           best = move;
76:         }
77:       }
78:     }
79:
80:     return bestMove;
81:   }
82: }
```

```
    1: package dev.gavinthomas.tictactoe;
    2:
    3: import dev.gavinthomas.tictactoe.Board.PieceType;
    4:
    5: import java.util.Arrays;
    6:
    7: public abstract class TTT {
    8:   public static PieceType getWinner(PieceType[][] board) {
    9:     return TTT.getWinner(board, board.length);
   10:   }
   11:   public static PieceType getWinner(PieceType[][] board, int size) {
   12:     // horizontals; x = column; y = row
   13:     for (int y = 0; y < size; y++) {
   14:       for (int x = 0; x < size; x++) {
   15:         if (x == 0) {
   16:           continue; // first value, so skip since nothing to compare to
   17:         }
   18:         if (board[x - 1][y] != board[x][y] || board[x][y] == PieceType.BLANK) {
   19:           break; // not all are same, or some are blank
   20:         }
   21:         if (x + 1 == size) {
   22:           return board[x][y]; // return the piece type since they would all matc
h according to check above
   23:         }
   24:       }
   25:     }
   26:
   27:     // verticals; x = column; y = row
   28:     for (int x = 0; x < size; x++) {
   29:       for (int y = 0; y < size; y++) {
   30:         if (y == 0) {
   31:           continue;
   32:         }
   33:         if (board[x][y - 1] != board[x][y] || board[x][y] == PieceType.BLANK) {
   34:           break;
   35:         }
   36:         if (y + 1 == size) {
   37:           return board[x][y];
   38:         }
   39:       }
   40:     }
   41:     // diagonals; xy = column & row
   42: //    if (size == 3) return null;
   43:
   44:     // bottom left to top right
   45:     for (int xy = 0; xy < size; xy++) {
   46:       if (xy == 0) {
   47:         continue;
   48:       }
   49:       if (board[xy - 1][xy - 1] != board[xy][xy] || board[xy][xy] == PieceType.B
LANK) {
   50:         break;
   51:       }
   52:       if (xy + 1 == size) {
   53:         return board[xy][xy];
   54:       }
   55:     }
   56:     // top left to bottom right; x = column; y = row
   57:     for (int x = 0; x < size; x++) {
   58:       int y = Math.abs(x - size + 1);
   59:       if (x == 0) {
   60:         continue;
   61:       }
```

```
62:       if (board[x - 1][y + 1] != board[x][y] || board[x][y] == PieceType.BLANK)
{
63:         break;
64:       }
65:       if (x + 1 == size) {
66:         return board[x][y];
67:       }
68:     }
69:
70:     return null;
71:   }
72:
73:   public static boolean gameOver(PieceType[][] board) {
74:     if (getWinner(board) != null) return true;
75:
76:     for (PieceType[] arr : board) {
77:       for (PieceType piece : arr) {
78:         if (piece == PieceType.BLANK) {
79:           return false;
80:         }
81:       }
82:     }
83:     return true;
84:   }
85: }
```

```java
package dev.gavinthomas.tictactoe.input;

import java.awt.Point;
import java.util.function.Consumer;
import java.util.Arrays;
import java.util.List;

import java.util.regex.Pattern;

public abstract class InputQueue {
  public static String codesToString(List<Integer> codes) {
    String str = "";
    for (int i = 0; i < codes.size(); i++) {
      str += Character.toString(codes.get(i));
    }
    return str;
  }

  public interface QueueType {
    public boolean matchStart(List<Integer> codes);

    public boolean matches(List<Integer> codes);

    public void run(List<Integer> codes);

    public int priority();

    public boolean doNext();
  }

  public static class TermSize implements QueueType {
    private final int PRIORITY;
    private final List<Object> ARGS;
    private final Consumer<Object[]> RUNNER;
    private final boolean DONEXT;

    private TermSize(Builder build) {
      this.PRIORITY = build.priority;
      this.ARGS = Arrays.asList(build.args);
      this.RUNNER = build.runner;
      this.DONEXT = build.doNext;
    }

    public boolean matchStart(List<Integer> codes) {
      String cstr = codesToString(codes);
      return pats.termSizePrefix.matcher(cstr).matches();
    }

    public boolean matches(List<Integer> codes) {
      String cstr = codesToString(codes);
      return pats.termSize.matcher(cstr).matches();
    }

    public void run(List<Integer> codes) {
      for (int i = 0; i < ARGS.size(); i++) {
        if (ARGS.get(i) == ArgType.CODES) {
          ARGS.set(i, codes);
        } else if (ARGS.get(i) == ArgType.TERMSIZE) {
          StringBuilder xVal = new StringBuilder();
          StringBuilder yVal = new StringBuilder();
          for (int j = 0; j < codes.size(); j++) {
            if (j <= 1 || codes.get(j) == 59) continue;
            if (codes.get(j) == 82) break;
```

```
    64:             if (xVal.length() != 0 || codes.get(j - 1) == 59) {
    65:               xVal.append((char) codes.get(j).intValue());
    66:             } else {
    67:               yVal.append((char) codes.get(j).intValue());
    68:             }
    69:           }
    70:           ARGS.set(i, new Point(Integer.parseInt(xVal.toString()), Integer.parse
Int(yVal.toString())));
    71:         }
    72:       }
    73:       RUNNER.accept(ARGS.toArray());
    74:     }
    75:
    76:     public int priority() {
    77:       return PRIORITY;
    78:     }
    79:
    80:     public boolean doNext() {
    81:       return DONEXT;
    82:     }
    83:
    84:     public static class Builder {
    85:       private int priority = 0;
    86:       private boolean doNext = false;
    87:       private Object[] args = new Object[0];
    88:       private Consumer<Object[]> runner;
    89:
    90:       public Builder(Consumer<Object[]> runner) {
    91:         this.runner = runner;
    92:       }
    93:
    94:       public Builder args(Object[] args) {
    95:         this.args = args;
    96:         return this;
    97:       }
    98:
    99:       public Builder priority(int priority) {
   100:         this.priority = priority;
   101:         return this;
   102:       }
   103:
   104:       public Builder doNext(boolean tog) {
   105:         this.doNext = tog;
   106:         return this;
   107:       }
   108:
   109:       public TermSize build() {
   110:         return new TermSize(this);
   111:       }
   112:     }
   113:   }
   114:
   115:   public static enum ArgType {
   116:     CODES, TERMSIZE;
   117:   }
   118:
   119:   private static abstract class pats {
   120:     public static Pattern termSize = Pattern.compile("\033\\[(\\d+)\\;(\\d+)R");
   121:     public static Pattern termSizePrefix = Pattern.compile(
   122:         "^((?:\033)|(?:\033\\[)|(?:\033\\[(?:\\d+))|(?:\033\\[(?:\\d+)\\;)|(?:\0
33\\[(?:\\d+)\\;(?:\\d+))|(?:\033\\[(?:\\d+)\\;(?:\\d+)R))$");
   123:   }
   124: }
```

```java
package dev.gavinthomas.tictactoe.input;

import java.util.function.Consumer;
import java.util.Arrays;
import java.util.List;
import java.util.ArrayList;

// import tetris.enums.KeyType;
// import tetris.utils.*;
// import tetris.enums.Command;
// import tetris.enums.KeybindArgument;
// import tetris.enums.Keycode;


public class Keybind {
  private final List<Object> ARGS;
  private final List<Keycode> KEYS;
  private final Consumer<Object[]> RUNNER;

  // new HashMap<String, Object>(Map.ofEntries(Map.entry("abc", 1), ...)); //
  // Unlimited
  // new HashMap<String, Object>(Map.of("abc", 1, ...)); // Up to 10

  public Keybind(Keycode key, Object[] args, Consumer<Object[]> runner) {
    this.KEYS = new ArrayList<Keycode>();
    this.KEYS.add(key);
    this.ARGS = replaceArgs(Arrays.asList(args));
    this.RUNNER = runner;
  }

  public Keybind(Keycode[] keys, Object[] args, Consumer<Object[]> runner) {
    this.KEYS = new ArrayList<Keycode>(Arrays.asList(keys));
    this.ARGS = replaceArgs(Arrays.asList(args));
    this.RUNNER = runner;
  }

  public Keybind(List<Keycode> keys, Object[] args, Consumer<Object[]> runner) {
    this.KEYS = new ArrayList<Keycode>(keys);
    this.ARGS = replaceArgs(Arrays.asList(args));
    this.RUNNER = runner;
  }

  public void addKeys(Keycode key) {
    this.KEYS.add(key);
  }

  public void addKeys(Keycode key, Keycode... keys) {
    this.KEYS.add(key);
    this.KEYS.addAll(Arrays.asList(keys));
  }

  public void removeKeys(Keycode key) {
    this.KEYS.remove(key);
  }

  public void removeKeys(Keycode key, Keycode... keys) {
    this.KEYS.remove(key);
    this.KEYS.removeAll(Arrays.asList(keys));
  }

  // public boolean[] hasKeys(int key) {

  // }
```

```
   64:
   65:   // public boolean[] hasKeys(int key, int... keys) {
   66:
   67:   // }
   68:
   69:   public boolean hasKey(Keycode key) {
   70:     return KEYS.contains(key);
   71:   }
   72:
   73:   public void run(Keycode key) {
   74:     List<Object> argCopy = new ArrayList<Object>(ARGS);
   75:     for (int i = 0; i < argCopy.size(); i++) {
   76:       if (argCopy.get(i) == KeybindArgument.KEYCODE) {
   77:         argCopy.set(i, key);
   78:       }
   79:     }
   80:     RUNNER.accept(argCopy.toArray());
   81:   }
   82:
   83:   public void handle(Keycode key) {
   84:     if (props.enabled && !props.forceDisabled && System.currentTimeMillis() > (p
rops.lastUsed + props.cooldown)) {
   85:       props.lastUsed = System.currentTimeMillis();
   86:       run(key);
   87:     }
   88:   }
   89:
   90:   private List<Object> replaceArgs(List<Object> args) {
   91:     for (int i = 0; i < args.size(); i++) {
   92:       if (args.get(i) == KeybindArgument.KEYBIND) {
   93:         args.set(i, this);
   94:       }
   95:     }
   96:     return args;
   97:   }
   98:
   99:   public final Getters get = new Getters();
  100:   public final Setters set = new Setters();
  101:   private final Properties props = new Properties();
  102:
  103:   private final class Properties {
  104:     private boolean enabled = true;
  105:     private boolean forceDisabled = false;
  106:     private int cooldown = 0;
  107:     private long lastUsed = 0;
  108:   }
  109:
  110:   public final class Getters {
  111:     public boolean enabled() {
  112:       return props.enabled;
  113:     }
  114:
  115:     public int cooldown() {
  116:       return props.cooldown;
  117:     }
  118:
  119:     public long lastUsed() {
  120:       return props.lastUsed;
  121:     }
  122:
  123:     public boolean forceDisabled() {
  124:       return props.forceDisabled;
  125:     }
```

```
126:   }
127:
128:   public final class Setters {
129:     public void enabled(boolean on) {
130:       RUNNER.accept(new Object[] { "abc" });
131:       props.enabled = on;
132:     }
133:
134:     public void cooldown(int time) {
135:       props.cooldown = time;
136:     }
137:
138:     public void lastUsed(long time) {
139:       props.lastUsed = time;
140:     }
141:
142:     public void forceDisabled(boolean tog) {
143:       props.forceDisabled = tog;
144:     }
145:   }
146: }
```

```
1: package dev.gavinthomas.tictactoe.input;
2:
3: public enum KeybindArgument {
4:   KEYBIND, KEYCODE;
5: }
```

```
    1: package dev.gavinthomas.tictactoe.input;
    2:
    3: import java.util.Arrays;
    4:
    5:
    6: public enum Keycode {
    7:
    8:   // lowercase letters
    9:   LOWER_A(97, 'a', "a"), LOWER_B(98, 'b', "b"), LOWER_C(99, 'c', "c"), LOWER_D(1
00, 'd', "d"), LOWER_E(101, 'e', "e"),
   10:   LOWER_F(102, 'f', "f"), LOWER_G(103, 'g', "g"), LOWER_H(104, 'h', "h"), LOWER_
I(105, 'i', "i"), LOWER_J(106, 'j', "j"),
   11:   LOWER_K(107, 'k', "k"), LOWER_L(108, 'l', "l"), LOWER_M(109,'m', "m"), LOWER_N
(110, 'n', "n"), LOWER_O(111, 'o', "o"),
   12:   LOWER_P(112, 'p', "p"), LOWER_Q(113, 'q', "q"), LOWER_R(114, 'r', "r"), LOWER_
S(115,'s', "s"), LOWER_T(116, 't', "t"),
   13:   LOWER_U(117, 'u', "u"), LOWER_V(118, 'v', "v"), LOWER_W(119, 'w', "w"), LOWER_
X(120, 'x', "x"), LOWER_Y(121, 'y', "y"),
   14:   LOWER_Z(122, 'z', "z"),
   15:
   16:   // uppercase letters
   17:   UPPER_A(65, 'A', "A"), UPPER_B(66, 'B', "B"), UPPER_C(67, 'C', "C"), UPPER_D(6
8, 'D', "D"), UPPER_E(69, 'E', "E"),
   18:   UPPER_F(70, 'F', "F"), UPPER_G(71, 'G', "G"), UPPER_H(72, 'H', "H"), UPPER_I(7
3, 'I', "I"), UPPER_J(74, 'J', "J"),
   19:   UPPER_K(75, 'K', "K"), UPPER_L(76, 'L', "L"), UPPER_M(77, 'M', "M"), UPPER_N(7
8, 'N', "N"), UPPER_O(79, 'O', "O"),
   20:   UPPER_P(80, 'P', "P"), UPPER_Q(81, 'Q', "Q"), UPPER_R(82, 'R', "R"), UPPER_S(8
3, 'S', "S"), UPPER_T(84, 'T', "T"),
   21:   UPPER_U(85, 'U', "U"), UPPER_V(86, 'V', "V"), UPPER_W(87, 'W', "W"), UPPER_X(8
8, 'X', "X"), UPPER_Y(89, 'Y', "Y"),
   22:   UPPER_Z(90, 'Z', "Z"),
   23:
   24:
   25:   // numbers
   26:   ZERO(48, '0', "0"), ONE(49, '1', "1"), TWO(50, '2', "2"), THREE(51, '3', "3"),
 FOUR(52, '4', "4"), FIVE(53, '5', "5"),
   27:   SIX(54, '6', "6"), SEVEN(55, '7', "7"), EIGHT(56, '8', "8"), NINE(57, '9', "9"
),
   28:
   29:   // special characters
   30:   PERIOD(46, '.', "."), COMMA(44, ',', ","), COLON(58, ':', ":"), SEMICOLON(59,
';', ";"), QUESTION_MARK(63, '?', "?"),
   31:   EXCLAMATION_POINT(33, '!', "!"), TILDE(126, '~', "~"), UNDERSCORE(95, '_', "_"
), MINUS(45, '-', "-"), PLUS(43, '+', "+"),
   32:   BACKSLASH(92, '\\', "\\"), FORWARD_SLASH(47, '/', "/"), GRAVE(96, '`', "`"), L
EFT_BRACKET(91, '[', "["),
   33:   RIGHT_BRACKET(93, ']', "]"), QUOTE(34, '"', "\""), APOSTROPHE(39, '\'', "\'"),
 LEFT_CURLY_BRACKET(123, '{', "{"),
   34:   RIGHT_CURLY_BRACKET(125, '}', "}"), LESS_THAN(60, '<', "<"), GREATER_THAN(62,
'>', ">"), EQUAL(61, '=', "="),
   35:
   36:
   37:   BACK_QUOTE(96, '`', "`"), VERTICAL_BAR(124, '|', "|"), AT(64, '@', "@"), POUND
(35, '#', "#"), CARET(94, '^', "^"),
   38:   AMPERSAND(38, '&', "&"), ASTERISK(42, '*', "*"), LEFT_PARENTHESIS(40, '(', "("
), RIGHT_PARENTHESIS(41, ')', ")"),
   39:   PERCENT(37, '%', "%"),
   40:
   41:   // mod keys
   42:   BACKSPACE(127, '\b', "Backspace"), SPACE(32, ' ', "Space"), TAB(9, '\t', "Tab"
), ENTER(13, '\n', "Enter"),
   43:
```

```
   44:   UP_ARROW(new int[]{27, 91, 65}, "Up Arrow"), DOWN_ARROW(new int[]{27, 91, 66},
 "Down Arrow"),
   45:   RIGHT_ARROW(new int[]{27, 91, 67}, "Right Arrow"), LEFT_ARROW(new int[]{27, 91
, 68}, "Left Arrow");
   46:
   47:
   48:   private int[] codes;
   49:   private Character key;
   50:   private String name;
   51:
   52:   Keycode(int code, Character key, String name) {
   53:     this.codes = new int[]{code};
   54:     this.key = key;
   55:     this.name = name;
   56:   }
   57:
   58:   Keycode(int[] codes, String name) {
   59:     this.codes = codes;
   60:     this.name = name;
   61:     this.key = null;
   62:   }
   63:
   64:   public int[] getCodes() {
   65:     return codes;
   66:   }
   67:   public Character getKey() {
   68:     return key;
   69:   }
   70:   public String getName() {
   71:     return name;
   72:   }
   73:
   74:
   75:   public static Keycode find(int code) {
   76:     for (Keycode key : Keycode.values()) {
   77:       if (key.getCodes()[0] == code) {
   78:         return key;
   79:       }
   80:     }
   81:     return null;
   82:   }
   83:
   84:   public static Keycode find(int[] codes) {
   85:     for (Keycode key : Keycode.values()) {
   86:       if (Arrays.equals(key.getCodes(), codes)) {
   87:         // System.out.println(key.getName());
   88:         return key;
   89:       }
   90:     }
   91:     return null;
   92:   }
   93:
   94:   public static Keycode find(Character keychar) {
   95:     for (Keycode key : Keycode.values()) {
   96:       if (key.getKey().equals(keychar)) {
   97:         return key;
   98:       }
   99:     }
  100:     return null;
  101:   }
  102:
  103:   public static Keycode find(String name) {
  104:         for (Keycode key : Keycode.values()) {
```

```
  105:                 if (key.getName().equals(name)) {
  106:                         return key;
  107:                 }
  108:         }
  109:         return null;
  110:   }
  111:
  112:   public static boolean hasNext(int[] codes) {
  113:     for (Keycode key : Keycode.values()) {
  114:       int[] next = key.getCodes();
  115:       if (codes.length < next.length && Arrays.equals(Arrays.copyOfRange(next, 0
, codes.length), codes)) {
  116:         // System.out.println(key);
  117:         return true;
  118:       }
  119:     }
  120:     return false;
  121:   }
  122:
  123:   public static Keycode matchStart(int[] codes) {
  124:     for (Keycode key : Keycode.values()) {
  125:       if (Arrays.equals(key.getCodes(), Arrays.copyOfRange(codes, 0, key.getCode
s().length))) {
  126:         // System.out.println(key.getName());
  127:         return key;
  128:       }
  129:     }
  130:     return null;
  131:   }
  132: }
```

```
    1: package dev.gavinthomas.tictactoe.input;
    2:
    3: import java.io.IOException;
    4: import java.util.*;
    5: import java.util.stream.IntStream;
    6: import java.util.concurrent.ExecutorService;
    7: import java.util.concurrent.Executors;
    8: import java.util.concurrent.RejectedExecutionException;
    9:
   10: import org.jline.terminal.*;
   11: import org.jline.utils.*;
   12: import org.jline.keymap.*;
   13:
   14: public abstract class input {
   15:   private static volatile boolean initiated = false;
   16:   private static volatile boolean readEnabled = false;
   17:   public static volatile List<InputQueue.QueueType> queue = new ArrayList<InputQ
ueue.QueueType>();
   18:   private static volatile ExecutorService exec = Executors.newSingleThreadExecut
or();
   19:
   20:   private static final List<Keybind> KEYS = new ArrayList<Keybind>();
   21:   private static final List<Integer> stream = new ArrayList<Integer>();
   22:   private static volatile Terminal terminal;
   23:   private static volatile NonBlockingReader reader;
   24:   private static volatile BindingReader bindReader;
   25:
   26:
   27:   public static void init() {
   28:     if (initiated) {
   29:       return;
   30:     }
   31:     initiated = true;
   32:     try {
   33:       terminal = TerminalBuilder.builder().jna(true).system(true).build();
   34:
   35:       terminal.enterRawMode();
   36:
   37:       reader = terminal.reader();
   38:       bindReader = new BindingReader(reader);
   39:     } catch (IOException e) {
   40:       e.printStackTrace();
   41:     }
   42:   }
   43:
   44:   public static void toggleRead(boolean tog) {
   45:     if (!readEnabled && tog) {
   46:       try {
   47:         exec.submit(input::read);
   48:       } catch (RejectedExecutionException e) {
   49:         exec = Executors.newSingleThreadExecutor();
   50:         exec.submit(input::read);
   51:       }
   52:       readEnabled = true;
   53:     } else if (readEnabled && !tog) {
   54:       readEnabled = false;
   55:       exec.shutdownNow();
   56:     }
   57:   }
   58:
   59:   public static void tread() {
   60:     while (true) {
   61:       while (readEnabled) {
```

```
   62:         Keycode key = nextKey();
   63:         if (!readEnabled) { // probably not needed when using shutdownNow since
everything is halted.
   64:           break;
   65:         }
   66:         if (key == null) {
   67:           continue;
   68:         }
   69:         for (Keybind kb : KEYS) {
   70:           if (kb.hasKey(key)) {
   71:             kb.handle(key);
   72:           }
   73:         }
   74:       }
   75:     }
   76:   }
   77:
   78:   public static void read() {
   79:     checkerloop: while (true) {
   80:       try {
   81:         stream.add(bindReader.readCharacter());
   82:         // Out.append(stream.size());
   83:         List<InputQueue.QueueType> queueCheck = new ArrayList<InputQueue.QueueTy
pe>(queue);
   84:         List<InputQueue.QueueType> matched = new ArrayList<InputQueue.QueueType>
();
   85: //       System.out.println(queueCheck.size());
   86:         for (int i = 0; i < queueCheck.size(); i++) {
   87: //         System.out.print(i + ", ");
   88:           if (queueCheck.get(i).matches(stream)) {
   89:             matched.add(queueCheck.get(i));
   90:           } else if (!queueCheck.get(i).matchStart(stream)) {
   91:             queueCheck.remove(i);
   92:           }
   93:         }
   94:
   95:         if (matched.size() != 0) {
   96:           matched.sort(new Comparator<InputQueue.QueueType>() {
   97:             public int compare(InputQueue.QueueType q1, InputQueue.QueueType q2)
 {
   98:               if (q1.priority() == q2.priority()) {
   99:                 return 0;
  100:               }
  101:               return (q1.priority() < q2.priority() ? -1 : 1);
  102:             }
  103:           });
  104:
  105:           for (int i = 0; i < matched.size(); i++) {
  106:             matched.get(i).run(stream);
  107:             queue.remove(matched.get(i));
  108:             if (matched.get(i).doNext()) {
  109:               continue;
  110:             } else {
  111:               stream.clear();
  112:               continue checkerloop;
  113:             }
  114:           }
  115:         }
  116:
  117:         if (queueCheck.size() > 0) continue;
  118:
  119:         Keycode matchCode = Keycode.find(stream.stream().mapToInt(Integer::intVa
lue).toArray());
```

```
  120:
  121:         if (matchCode == null && !Keycode.hasNext(stream.stream().mapToInt(Integ
er::intValue).toArray())) {
  122:           stream.clear();
  123:           continue;
  124:         }
  125:         try {
  126:           for (Keybind kb : KEYS) {
  127:             if (kb.hasKey(matchCode)) {
  128:               kb.handle(matchCode);
  129:             }
  130:           }
  131:         } catch (ConcurrentModificationException ignore) {}
  132:         if (!Keycode.hasNext(stream.stream().mapToInt(Integer::intValue).toArray
())) {
  133:           stream.clear();
  134:         }
  135:       } catch (Exception e) {
  136:         e.printStackTrace();
  137:       }
  138:     }
  139:   }
  140:
  141:
  142:   private static int checkNull(List<InputQueue.QueueType> qms) {
  143:     int count = 0;
  144:     for (int i = 0; i < qms.size(); i++) {
  145:       count += (qms.get(i) != null ? 1 : 0);
  146:     }
  147:     return count;
  148:   }
  149:
  150:   public static Keycode nextKey() {
  151:     List<Integer> codes = new ArrayList<Integer>();
  152:     List<InputQueue.QueueType> qms = new ArrayList<InputQueue.QueueType>(queue);
  153:
  154:     // codes.add(bindReader.readCharacter());
  155:
  156:
  157:
  158:     checkerloop: while (checkNull(qms) > 0 || Keycode.hasNext(codes.stream().map
ToInt(Integer::intValue).toArray())) {
  159:       List<InputQueue.QueueType> matched = new ArrayList<InputQueue.QueueType>()
;
  160:               // System.out.println("checkLoop"); // ..............
  161:
  162:       codes.add(bindReader.readCharacter());
  163:       for (int i = 0; i < qms.size(); i++) {
  164:         if (qms.get(i) == null) {
  165:           qms.remove(i);
  166:           continue;
  167:         }
  168:
  169:         if (qms.get(i).matches(codes)) {
  170:           matched.add(qms.get(i));
  171:         } else if (!qms.get(i).matchStart(codes)) {
  172:           qms.remove(i);
  173:         }
  174:       }
  175:
  176:
  177:       if (matched.size() == 0) {
  178:         continue;
```

```
  179:       }
  180:       Collections.sort(matched, new Comparator<InputQueue.QueueType>() {
  181:         public int compare(InputQueue.QueueType q1, InputQueue.QueueType q2) {
  182:           if (q1.priority() == q2.priority()) {
  183:             return 0;
  184:           }
  185:           return (q1.priority() < q2.priority() ? -1 : 1);
  186:         }
  187:       });
  188:
  189:       for (int i = 0; i < matched.size(); i++) {
  190:         System.out.println(i);
  191:         matched.get(i).run(codes);
  192:         queue.remove(matched.get(i));
  193:         if (matched.get(i).doNext()) {
  194:           continue;
  195:         } else {
  196:           // break checkerloop; // Should break this loop as well. Check later.
  197:           return null;
  198:         }
  199:       }
  200:       // codes.add(bindReader.readCharacter());
  201:     }
  202:     while (Keycode.hasNext(codes.stream().mapToInt(Integer::intValue).toArray())
) {
  203:       // System.out.println(bindReader.readCharacter());
  204:       codes.add(bindReader.readCharacter());
  205:     }
  206:
  207:     return Keycode.find(codes.stream().mapToInt(Integer::intValue).toArray());
  208:   }
  209:
  210: //  public static void getSizeReport() {
  211: //    System.out.print("\033[s\033[50000;50000H\033[6n\033[u");
  212: //    awaitingSizeReport = true;
  213: //  }
  214:
  215:   public static void addBinds(Keybind kb) {
  216:     KEYS.add(kb);
  217:   }
  218:
  219:   public static void addBinds(Keybind kb, Keybind... kbs) {
  220:     KEYS.add(kb);
  221:     KEYS.addAll(Arrays.asList(kbs));
  222:   }
  223:
  224:   public static void addBinds(List<Keybind> kbs) {
  225:     KEYS.addAll(kbs);
  226:   }
  227:
  228:   public static void removeBinds(Keybind kb) {
  229:     KEYS.remove(kb);
  230:   }
  231:
  232:   public static void removeBinds(Keybind kb, Keybind... kbs) {
  233:     KEYS.remove(kb);
  234:     KEYS.removeAll(Arrays.asList(kbs));
  235:   }
  236:
  237:   public static void removeBinds(List<Keybind> kbs) {
  238:     KEYS.removeAll(kbs);
  239:   }
  240:
```

```
241:   public static void clearBinds() {
242:     KEYS.clear();
243:   }
244:
245:   public static int nextChar() {
246:     return bindReader.readCharacter();
247:   }
248:
249:   public static boolean arrContainsFinal(int[] arr, int key) {
250:     final int val = key;
251:     return IntStream.of(arr).anyMatch(i -> i == val);
252:   }
253:
254:   public static int awaitKey(int[] validKeys) {
255:     int currKey = nextChar();
256:     while (arrContainsFinal(validKeys, currKey) == false) {
257:       System.out.println(currKey + " | " + (char) currKey);
258:       currKey = nextChar();
259:     }
260:     return currKey;
261:   }
262: }
```

```java
package dev.gavinthomas.tictactoe.ui;

import dev.gavinthomas.tictactoe.TicTacToe;
import dev.gavinthomas.tictactoe.input.*;
import dev.gavinthomas.tictactoe.types.UIComponent;
import dev.gavinthomas.tictactoe.types.UIHolder;
import dev.gavinthomas.tictactoe.types.Visuals;
import dev.gavinthomas.tictactoe.utils.Term;

import dev.gavinthomas.tictactoe.ui.SelectionUI.Selection;
import dev.gavinthomas.tictactoe.utils.listeners;

import java.awt.Point;
import java.util.ArrayList;
import java.util.List;

public class Menu implements UIHolder {
  private final Term TERM = TicTacToe.CURR.TERM;
  private final List<UIComponent> comps = new ArrayList<UIComponent>();
  private final List<Keybind> KBS = new ArrayList<Keybind>();
  private Point offset;
  private final Point size = new Point(90, 30);

  public Menu() {
    Selection[] sArr = {
        new Selection("New Game", this::temp, new Object[]{}),
        new Selection("Load Game", this::temp, new Object[]{}),
        new Selection("Settings", this::temp, new Object[]{})
    };

//    comps.get(0).render();
    SelectionUI sui = new SelectionUI(this, sArr, new Point((size().x / 2) - 10,
 10));
    comps.add(sui);
    comps.add(new Title(this, new Point((size().x / 2) - 39, 1)));
    KBS.add(new Keybind(
        new Keycode[] { Keycode.UP_ARROW },
        new Object[] { KeybindArgument.KEYCODE }, sui::moveUp));
    KBS.add(new Keybind(
        new Keycode[] { Keycode.DOWN_ARROW },
        new Object[] { KeybindArgument.KEYCODE }, sui::moveDown));
    KBS.add(new Keybind(
        new Keycode[] { Keycode.SPACE },
        new Object[] { KeybindArgument.KEYCODE }, sui::select));
    TicTacToe.CURR.registerKB(KBS);
  }

  public void endTasks() {
    for (UIComponent comp : comps) {
      comp.endTasks();
    }
  }

  public void render() {
    for (UIComponent comp : comps) {
      comp.endTasks();
    }
    TERM.clear(true);
    TERM.hideCursor(true);
    Point tSize = TicTacToe.CURR.SIZE;
//    offset = new Point((tSize.x / 2) - (size().x / 2), (tSize.y / 2) - (size()
.y / 2));
    offset = new Point((tSize.x / 2) - (size().x / 2), 1);
```

```
62:     TERM.setCursorPos(offset.x, offset.y);
63:     TERM.print(Visuals.doubleLineBox(size.x, size.y));
64:
65:     for (UIComponent comp : comps) {
66:       comp.render();
67:     }
68:
69: //    System.out.print(tempFill.repeat(15));
70:
71: //    comps.get(0).render();
72:   }
73:
74:   public void temp(Object[] args) {
75:
76:   }
77:
78:   public Point offset() {
79: //    return new Point(offset.x + 2, offset.y + 1);
80:     return new Point(offset.x + 1, offset.y + 1);
81:   }
82:
83:   public Point size() {
84: //    return new Point(size.x - 4, size.y - 2);
85:     return new Point(size.x - 2, size.y - 2);
86:   }
87:
88:   public void setLocation(Point pos, String str) {
89:     TERM.saveCursor();
90:     TERM.setCursorPos(pos.x, pos.y);
91:     System.out.print(str);
92:     TERM.restoreCursor();
93:   }
94: }
```

```
    1: package dev.gavinthomas.tictactoe.ui;
    2:
    3: import dev.gavinthomas.tictactoe.TicTacToe;
    4: import dev.gavinthomas.tictactoe.types.UIComponent;
    5: import dev.gavinthomas.tictactoe.types.UIHolder;
    6: import dev.gavinthomas.tictactoe.types.Visuals;
    7: import dev.gavinthomas.tictactoe.utils.Term;
    8:
    9: import java.awt.Point;
   10: import java.util.Arrays;
   11: import java.util.List;
   12: import java.util.function.Consumer;
   13: import java.util.function.Function;
   14:
   15: public class SelectionUI implements UIComponent {
   16:   private final Term TERM = TicTacToe.CURR.TERM;
   17:   public static final String[] uiFormatting = {"\033[1m> ", " <\033[0m"};
   18:   private Selection selected;
   19:   private Point pos;
   20:   private final UIHolder holder;
   21:   public final List<Selection> selections;
   22:   java.util.concurrent.Semaphore s = new java.util.concurrent.Semaphore(0);
   23:
   24:   public SelectionUI(UIHolder holder, Selection[] selections, Point pos) {
   25:     this.selections = Arrays.asList(selections);
   26:     this.pos = pos;
   27:     this.holder = holder;
   28: //    this.setSelected(this.selections.get(findNotDisabled(-1, true)));
   29:   }
   30:
   31:   public void render() {
   32:     setCursorPos(pos.x, pos.y);
   33:     TERM.print(getRender());
   34:     if (selected == null) {
   35:       setSelected(selections.get(findNotDisabled(-1, true)));
   36:     }
   37:   }
   38:
   39:   public void endTasks() {}
   40:
   41:   public String getRender() {
   42:     StringBuilder vals = new StringBuilder();
   43:     for (Selection s : selections) {
   44:       vals.append(SelectionUI.getComp(s.NAME, s.mode, selections.indexOf(s) != s
elections.size() - 1));
   45:     }
   46:     return vals.toString();
   47:   }
   48:
   49:   public void disable(Selection val, boolean tog) {
   50:     if (selected != val) {
   51:       val.mode = (tog ? MODE.DISABLED : MODE.NORMAL);
   52:       return;
   53:     }
   54:     int thisIndex = selections.indexOf(val);
   55:     int newIndex = (findNotDisabled(thisIndex, false) != -1 ?
   56:         findNotDisabled(thisIndex, false) :
   57:         findNotDisabled(thisIndex, true));
   58:     val.mode = MODE.DISABLED;
   59:     setSelected(selections.get(newIndex));
   60:   }
   61:
   62:   public void setCursorPos(int x, int y) {
```

```
    63:     TERM.setCursorPos(holder.offset().x + x, holder.offset().y + y);
    64:   }
    65:
    66:   public void setSelected(Selection newSelect) {
    67:     if (selected != null) {
    68:       setCursorPos(pos.x, pos.y + (selections.indexOf(selected) * 3));
    69:       TERM.print(SelectionUI.getComp(selected.NAME,
    70:           (selected.mode == MODE.SELECTED ? MODE.NORMAL : MODE.DISABLED),
    71:           false));
    72:       selected.mode = (selected.mode == MODE.SELECTED ? MODE.NORMAL : selected.m
ode);
    73:     }
    74:     this.selected = newSelect;
    75:     setCursorPos(pos.x, pos.y + (selections.indexOf(selected) * 3));
    76:     TERM.print(SelectionUI.getComp(selected.NAME, MODE.SELECTED, false));
    77:     newSelect.mode = MODE.SELECTED;
    78:   }
    79:
    80:   public void moveUp(Object[] args) {
    81:     int sInd = selections.indexOf(selected);
    82:     int next = findNotDisabled(selections.indexOf(selected), false);
    83:     if (next == -1) return;
    84:     setSelected(selections.get(next));
    85:   }
    86:
    87:   public void moveDown(Object[] args) {
    88:     int sInd = selections.indexOf(selected);
    89:     int next = findNotDisabled(selections.indexOf(selected), true);
    90:     if (next == -1) return;
    91:     setSelected(selections.get(next));
    92:   }
    93:
    94:   public void select(Object[] args) {
    95:     if (selected == null) return;
    96:     selected.RUNNER.accept(selected.args);
    97:   }
    98:
    99:   private int findNotDisabled(int start, boolean posIncrement) {
   100:     if (posIncrement) {
   101:       for (int i = start + 1; i < selections.size(); i++) {
   102:         if (selections.get(i).mode != MODE.DISABLED) return i;
   103:       }
   104:     } else {
   105:       for (int i = start - 1; i >= 0; i--) {
   106:         if (selections.get(i).mode != MODE.DISABLED) return i;
   107:       }
   108:     }
   109:     return -1;
   110:   }
   111:
   112:
   113:   public static class Selection {
   114:     public final String NAME;
   115:     public final Consumer<Object[]> RUNNER;
   116:     public boolean disabled = false;
   117:     public MODE mode = MODE.NORMAL;
   118:     public Object[] args;
   119:
   120:     public Selection(String name, Consumer<Object[]> runner, Object[] args) {
   121:       this.NAME = name;
   122:       this.RUNNER = runner;
   123:       this.args = args;
   124:     }
```

```
  125:
  126:     public void run() {
  127:       this.RUNNER.accept(args);
  128:     }
  129:   }
  130:
  131:   public static String getComp(String name, MODE mode, boolean cursorMove) {
  132:     return Visuals.menuButton(name, mode) + (cursorMove ? "\033[1B\033[20D" : ""
);
  133:   }
  134:
  135:   public enum MODE {
  136:     DISABLED, NORMAL, SELECTED
  137:   }
  138: }
```

```java
package dev.gavinthomas.tictactoe.ui;

import dev.gavinthomas.tictactoe.TicTacToe;
import dev.gavinthomas.tictactoe.types.UIComponent;
import dev.gavinthomas.tictactoe.types.UIHolder;
import dev.gavinthomas.tictactoe.types.Visuals;
import dev.gavinthomas.tictactoe.utils.Term;

import java.awt.Point;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.RejectedExecutionException;
import java.util.function.BiConsumer;

public class Title implements UIComponent {
  private final Term TERM = TicTacToe.CURR.TERM;
  private final Point pos;
  private final UIHolder holder;
  private volatile long animStart = 0;
  private volatile boolean animRunning = false;
  private ExecutorService animExec;
//  private final AnimProps[] anims = {
//      new AnimProps(50, 1000, this::titleAnim1),
//      new AnimProps(250, 2000, this::titleAnim2),
//  };
  private final AnimProps[] anims = {
      new AnimProps(50, 1, this::titleAnim1),
      new AnimProps(250, 6, this::titleAnim2),
  };

  public Title(UIHolder holder, Point pos) {
    this.holder = holder;
    this.pos = pos;
    this.animExec = Executors.newSingleThreadExecutor();
  }

  public void render() {
    toggleAnimations(false);
    setCursorPos(pos.x, pos.y);
    TERM.print(Visuals.title(new boolean[]{false, false, false, false, false, false, false, false, false}));
    toggleAnimations(true);
  }

  public void toggleAnimations(boolean tog) {
    if (tog) {
      if (animStart != 0) animExec.shutdownNow();
      try {
        animExec.submit(this::startAnimations);
      } catch (RejectedExecutionException e) {
        animExec = Executors.newSingleThreadExecutor();
        animExec.submit(this::startAnimations);
      }
    } else if (animStart != 0) {
      animStart = 0;
      animExec.shutdownNow();
    }
  }

  public void endTasks() {
    toggleAnimations(false);
  }

```

```
   63:   private void startAnimations() {
   64:     long selfStartTime = System.currentTimeMillis();
   65:     Point lastSize = TicTacToe.CURR.SIZE;
   66:     animStart = selfStartTime;
   67:
   68:     try {
   69:       Thread.sleep(1000);
   70:     } catch (InterruptedException ignored) {}
   71:     if (TicTacToe.CURR.SIZE != lastSize) return;
   72:
   73:     AnimProps current = null;
   74:
   75:     while (animStart == selfStartTime) {
   76:       AnimProps rand;
   77:       do {
   78:         rand = anims[(int) (Math.random() * anims.length)];
   79:       } while (rand == current);
   80:       current = rand;
   81:       rand.RUNNER.accept(rand.DELAY, rand.DURATION);
   82:     }
   83:   }
   84:
   85:   public void setCursorPos(int x, int y) {
   86:     TERM.setCursorPos(holder.offset().x + x, holder.offset().y + y);
   87:   }
   88:
   89:   private void titleAnim1(Integer delay, Integer duration) {
   90:     long startTime = System.currentTimeMillis();
   91: //    while (System.currentTimeMillis() - startTime < duration) {
   92:     for (int rep = 1; rep <= duration; rep++) {
   93:       boolean[] currOn = {false, false, false, false, false, false, false, false
, false};
   94:       int step = -2;
   95:       while (step < 10) {
   96: //        if (System.currentTimeMillis() - startTime < duration) break;
   97:         for (int i = step; i < step + 3; i++) {
   98:           if (i > 8 || i < 0) continue;
   99:           currOn[i] = true;
  100:         }
  101:         setCursorPos(pos.x, pos.y);
  102:         TERM.print(Visuals.title(currOn));
  103:         step++;
  104:         currOn = new boolean[]{false, false, false, false, false, false, false,
false, false};
  105:         try {
  106:           Thread.sleep(delay);
  107:         } catch (InterruptedException ignore) {
  108:         }
  109:       }
  110:       step = 9;
  111:       while (step > -2) {
  112: //        if (System.currentTimeMillis() - startTime < duration) break;
  113:         for (int i = step; i > step - 3; i--) {
  114:           if (i > 8 || i < 0) continue;
  115:           currOn[i] = true;
  116:         }
  117:         setCursorPos(pos.x, pos.y);
  118:         TERM.print(Visuals.title(currOn));
  119:         step--;
  120:         currOn = new boolean[]{false, false, false, false, false, false, false,
false, false};
  121:         try {
  122:           Thread.sleep(delay);
```

```
  123:         } catch (InterruptedException ignore) {
  124:         }
  125:       }
  126:     }
  127:     setCursorPos(pos.x, pos.y);
  128:     TERM.print(Visuals.title(new boolean[]{false, false, false, false, false, fa
lse, false, false, false}));
  129:   }
  130:
  131:   private void titleAnim2(Integer delay, Integer duration) {
  132:     long startTime = System.currentTimeMillis();
  133:     int i = 0;
  134: //    while(System.currentTimeMillis() - startTime < duration) {
  135:     for (int rep = 1; rep <= duration; rep++) {
  136:       boolean[] currOn = (i % 2 == 0 ?
  137:           new boolean[]{true, false, true, false, true, false, true, false, true
} :
  138:           new boolean[]{false, true, false, true, false, true, false, true, fals
e});
  139:       setCursorPos(pos.x, pos.y);
  140:       TERM.print(Visuals.title(currOn));
  141:       try {
  142:         Thread.sleep(delay);
  143:       } catch (InterruptedException ignore) {
  144:       }
  145:       i++;
  146:     }
  147:     setCursorPos(pos.x, pos.y);
  148:     TERM.print(Visuals.title(new boolean[]{false, false, false, false, false, fa
lse, false, false, false}));
  149:   }
  150: }
  151:
  152: class AnimProps {
  153:   public final Integer DELAY, DURATION;
  154:   public final BiConsumer<Integer, Integer> RUNNER;
  155:   public AnimProps(int DELAY, int DURATION, BiConsumer<Integer, Integer> RUNNER)
 {
  156:     this.DELAY = DELAY;
  157:     this.DURATION = DURATION;
  158:     this.RUNNER = RUNNER;
  159:   }
  160: }
```

```
    1: package dev.gavinthomas.tictactoe;
    2:
    3: import java.awt.Point;
    4: import java.util.ArrayList;
    5: import java.util.Arrays;
    6: import java.util.List;
    7:
    8: import dev.gavinthomas.tictactoe.types.UIHolder;
    9: import dev.gavinthomas.tictactoe.types.Visuals;
   10: import dev.gavinthomas.tictactoe.utils.Term;
   11:
   12: // 100x34 min size
   13: public class Board implements UIHolder {
   14:   private final Term TERM = TicTacToe.CURR.TERM;
   15:   private Point offset;
   16:   private PieceType[] plrs = new PieceType[2];
   17:   public final PieceType[][] grid = new PieceType[3][3];
   18:   public volatile boolean tempGetCompMove = false;
   19:   private final Point brdOffset = new Point(21, 0);
   20:   private final List<Point> highlights = new ArrayList<Point>();
   21:
   22:   public Board(int xPlr) {
   23:     this.plrs[xPlr] = PieceType.X;
   24:     this.plrs[(xPlr == 0 ? 1 : 0)] = PieceType.O;
   25:
   26:     for (PieceType[] arr : grid) {
   27:       Arrays.fill(arr, PieceType.BLANK);
   28:     }
   29:   }
   30:
   31:   public Point offset() {
   32:     return new Point();
   33:   }
   34:
   35:   public Point size() {
   36:     return new Point();
   37:   }
   38:
   39:   public void setPiece(int x, int y, PieceType piece) {
   40:     if (TicTacToe.CURR.invalidSize) return;
   41:     TERM.saveCursor();
   42:     TERM.setCursorPos(TicTacToe.OFFSET.x + brdOffset.x + (5 + (x * 20)),
   43:         TicTacToe.OFFSET.y + (2 + (Math.abs(y - (grid.length - 1)) * 10)));
   44:
   45:     if (piece == PieceType.X) {
   46:       System.out.print(Visuals.XBLOCK);
   47:     } else if (piece == PieceType.O) {
   48:       System.out.print(Visuals.OBLOCK);
   49:     } else {
   50:       System.out.print(Visuals.BLANKBLOCK);
   51:     }
   52:     TERM.restoreCursor();
   53:   }
   54:
   55:   public void highlightSpot(int x, int y, boolean tog) {
   56:     if (tog && highlights.stream().noneMatch(pt -> pt.equals(new Point(x, y))))
{
   57:       highlights.add(new Point(x, y));
   58:     } else if (!tog) {
   59:       highlights.removeIf(pt -> pt.equals(new Point(x, y)));
   60:     }
   61:     if (TicTacToe.CURR.invalidSize) return;
   62:     TERM.saveCursor();
```

```
63:     TERM.setCursorPos(TicTacToe.OFFSET.x + brdOffset.x + (1 + (x * 20)),
64:         TicTacToe.OFFSET.y + (1 + (Math.abs(y - (grid.length - 1))
65:             * 10)));
66:
67:     if (tog) {
68:       System.out.print(Visuals.HIGHLIGHT);
69:     } else {
70:       System.out.print(Visuals.UNHIGHLIGHT);
71:     }
72:
73:
74:     TERM.restoreCursor();
75:   }
76:
77:   public void render() {
78:     if (TicTacToe.CURR.invalidSize) return;
79:     TERM.clear(true);
80:     TERM.hideCursor(true);
81:     System.out.println("\n".repeat(TicTacToe.OFFSET.y) +
82:         " ".repeat(TicTacToe.OFFSET.x) + " ".repeat(brdOffset.x) +
83:         Visuals.GRID + "\n\n");
84:
85:     for (int i = 0; i < grid.length; i++) {
86:       for (int j = 0; j < grid[i].length; j++) {
87:         setPiece(i, j, grid[i][j]);
88:       }
89:     }
90:     for (int i = 0; i < highlights.size(); i++) {
91:       Point pt = highlights.get(i);
92:       highlights.remove(i);
93:       highlightSpot(pt.x, pt.y, true);
94:     }
95:
96: //    highlightSpot(0, 0, true);
97:   }
98:
99:   public PieceType getPiece(int x, int y) {
100:     if (x > 2 || x < 0 || y > 2 || y < 0) return null;
101:     return grid[x][y];
102:   }
103:
104:   public enum PieceType {
105:     X, O, BLANK
106:   }
107: }
```

```
    1: package dev.gavinthomas.tictactoe;
    2:
    3: import dev.gavinthomas.tictactoe.input.Keybind;
    4: import dev.gavinthomas.tictactoe.input.KeybindArgument;
    5: import dev.gavinthomas.tictactoe.input.Keycode;
    6: import dev.gavinthomas.tictactoe.input.input;
    7: import dev.gavinthomas.tictactoe.opponents.Computer;
    8: import dev.gavinthomas.tictactoe.opponents.Player;
    9: import dev.gavinthomas.tictactoe.types.Opponent;
   10: import dev.gavinthomas.tictactoe.Board;
   11: import dev.gavinthomas.tictactoe.Board.PieceType;
   12: import dev.gavinthomas.tictactoe.TTT;
   13:
   14: import java.awt.Point;
   15: import java.util.ArrayList;
   16: import java.util.List;
   17:
   18: public class Game {
   19:   private final Board board;
   20:   private final Opponent[] plrs = new Opponent[2];
   21:   private final List<Keybind> KBS = new ArrayList<Keybind>();
   22:   private int currentTurn;
   23:   private volatile boolean finished = false;
   24:   private Keybind inputKB;
   25:
   26:   public Game(Builder config) {
   27:     this.board = new Board(config.firstMove);
   28:     currentTurn = config.firstMove;
   29:     plrs[0] = new Player(this.board, (config.firstMove == 0 ? PieceType.X : Piec
eType.O), this::handleMove);
   30:     if (config.computer) {
   31:       plrs[1] = new Computer(this.board, (config.firstMove == 1 ? PieceType.X :
PieceType.O), this::handleMove);
   32:     } else {
   33:       plrs[1] = new Player(this.board, (config.firstMove == 1 ? PieceType.X : Pi
eceType.O), this::handleMove);
   34:     }
   35:   }
   36:
   37:   public void start() {
   38:     KBS.add(new Keybind(
   39:         new Keycode[] { Keycode.UP_ARROW, Keycode.DOWN_ARROW, Keycode.LEFT_ARROW
, Keycode.RIGHT_ARROW, Keycode.SPACE, Keycode.LOWER_L },
   40:         new Object[] { KeybindArgument.KEYCODE }, ((Player) plrs[0])::handleInpu
t));
   41:     if (plrs[1] instanceof Player) {
   42:       KBS.add(new Keybind(
   43:           new Keycode[] { Keycode.UP_ARROW, Keycode.DOWN_ARROW, Keycode.LEFT_ARR
OW, Keycode.RIGHT_ARROW, Keycode.SPACE, Keycode.LOWER_L },
   44:           new Object[] { KeybindArgument.KEYCODE }, ((Player) plrs[1])::handleIn
put));
   45:     }
   46:     this.board.render();
   47: //    input.toggleRead(true);
   48:
   49: //    input.addBinds(KBS);
   50:     TicTacToe.CURR.registerKB(KBS);
   51:     plrs[currentTurn].getMove();
   52: //    while (!TTT.gameOver(board.grid)) {
   53:     while (!finished) {
   54:       Thread.onSpinWait();
   55:     }
   56:
```

```
57: //    System.out.println(TTT.getWinner(board.grid));
58:
59: //    input.removeBinds(KBS);
60:     TicTacToe.CURR.deregisterKB(KBS);
61:   }
62:
63:   public void render() {
64:     this.board.render();
65:   }
66:
67:   public void handleMove(Point pt) {
68:     board.setPiece(pt.x, pt.y, plrs[currentTurn].getPiece());
69:     board.grid[pt.x][pt.y] = plrs[currentTurn].getPiece();
70: //    System.out.println(TTT.gameOver(board.grid));
71:     if (TTT.gameOver(board.grid)) {
72:       finished = true;
73:       return;
74:     }
75:     currentTurn = (currentTurn == 0 ? 1 : 0);
76:     if (plrs[1] instanceof Player) {
77:       ((Player) plrs[currentTurn]).getMove(pt.x, pt.y);
78:     } else {
79:       plrs[currentTurn].getMove();
80:     }
81:   }
82:
83:
84:   public void input(Object[] args) {
85:     if (plrs[currentTurn] instanceof Player tempPlr) {
86:       tempPlr.handleInput(args);
87:     }
88:   }
89:
90:   public static class Builder {
91:     private boolean computer = true; // playing against computer
92:     private int firstMove = (int) (2 * Math.random());
93:
94:     public Builder computerOpponent() {
95:       this.computer = true;
96:       return this;
97:     }
98:
99:     public Builder playerOpponent() {
100:      this.computer = false;
101:      return this;
102:    }
103:
104:    public Builder firstMove(int plr) {
105:      if (plr != 0 && plr != 1)
106:        return this;
107:      this.firstMove = plr;
108:      return this;
109:    }
110:
111:    public Game build() {
112:      return new Game(this);
113:    }
114:  }
115: }
```

```java
package dev.gavinthomas.tictactoe;

import dev.gavinthomas.tictactoe.input.input;
import dev.gavinthomas.tictactoe.types.Visuals;
import dev.gavinthomas.tictactoe.Board;
import dev.gavinthomas.tictactoe.Board.PieceType;
import dev.gavinthomas.tictactoe.TicTacToe;

import dev.gavinthomas.tictactoe.TTT;
import dev.gavinthomas.tictactoe.ui.Menu;
import dev.gavinthomas.tictactoe.utils.Minimax;
import dev.gavinthomas.tictactoe.utils.Term;
import io.raffi.drawille.Canvas;
import io.raffi.drawille.Turtle;

import java.awt.Point;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;


public class Main {
  public static final PieceType X = PieceType.X;
  public static final PieceType O = PieceType.O;
  public static boolean abc = true;
  public static final Term TERM = new Term();

  public static void main(String[] args) throws Exception {

//    System.out.println(Visuals.title1(new boolean[]{true, true, true}));

//System.out.println(Visuals.box(40, 20));
    TicTacToe tttG = new TicTacToe();
    tttG.init();
    while (abc) {
//      Thread.onSpinWait();
      tttG.newGame(new Game.Builder().playerOpponent().firstMove(0));
    }

//    Game g = new Game.Builder().computerOpponent().firstMove(0).build();
//    g.start();


//    Turtle turtle = new Turtle ( 75, 50 );
//    turtle.move ( turtle.getWidth () / 2, turtle.getHeight () / 2 );
//    turtle.down ();
//    for ( int x = 0; x < 72; x++ ) {
//      turtle.right ( 20 );
//      for ( int y = 0; y < 72; y++ ) {
//        turtle.right ( 20 );
//        turtle.forward ( 10 );
//      }
//    }
//    turtle.render ();
//    System.out.println(Visuals.TITLE);
      //    new Menu().render();
      //    input.toggleRead(true);
      //    while (Main.abc) {
      //      Thread.onSpinWait();
      //    }
    // System.out.print("\033[2J\033[H");

//     Board b = new Board(1);
```

```
   64: //     TicTacToe t = new TicTacToe();
   65: //     b.render();
   66: //     t.newGame(new Game.Builder().computerOpponent().firstMove(0));
   67:
   68:     PieceType[][] pta = new PieceType[4][4];
   69:     for (PieceType[] pieceTypes : pta) {
   70:       Arrays.fill(pieceTypes, PieceType.BLANK);
   71:     }
   72:
   73:     // pta[4][0] = X;
   74:     // pta[3][1] = X;
   75:     // pta[2][2] = X;
   76:     // pta[1][3] = X;
   77:     // pta[0][4] = X;
   78:
   79:     // pta[0][0] = X;
   80:     // pta[1][1] = X;
   81:     // pta[2][2] = X;
   82:     // pta[3][3] = X;
   83:     // pta[4][4] = X;
   84:
   85: //    pta[0][0] = X;
   86: //    pta[1][0] = X;
   87: //    pta[2][0] = O;
   88:
   89:     String[] brd = {
   90:         "O - X O",
   91:         "X - O O",
   92:         "X O X X",
   93:         "X - - O",
   94:     };
   95: //    String[] brd = {
   96: //        "OXO",
   97: //        "XOX",
   98: //        "XOX",
   99: //    };
  100:
  101:     for (int i = 0; i < brd.length; i++) {
  102:       for (int j = 0; j < brd[i].length(); j += 2) {
  103:         if (brd[i].charAt(j) == '-') continue;
  104: //        pta[Math.abs(i - brd.length + 1)][j] = (brd[i].charAt(j) == 'X' ? X :
O);
  105:         pta[j / 2][Math.abs(i - brd.length + 1)] = (brd[i].charAt(j) == 'X' ? X
: O);
  106:       }
  107:     }
  108: //    pta[0][1] = O;
  109: //    pta[1][1] = O;
  110: //    pta[2][1] = O;
  111: //    pta[2][2] = X;
  112:
  113:     // pta[0][0] = X;
  114:     // pta[1][0] = X;
  115:     // pta[2][0] = X;
  116:     // pta[3][0] = X;
  117:     // pta[4][0] = X;
  118:     long ts = System.currentTimeMillis();
  119: //    System.out.println(TTT.getWinner(pta));
  120:     Point rval = new Minimax(O).getBest(pta);
  121:     System.out.println("----------------------");
  122:     System.out.println(rval.x + ", " + rval.y);
  123:     System.out.println(new Minimax(O));
  124:     System.out.println(System.currentTimeMillis() - ts);
```

```
125:     // // String temp = """
126:     // //   abct
127: //    // // """;
128: //    String temp = "\n";
129: //    System.out.println(((int) temp.charAt(0)));
130:
131: //    new Scanner(System.in).nextLine(); // no auto exit
132:   }
133: }
```

```
    1: package dev.gavinthomas.tictactoe;
    2:
    3: import java.awt.Point;
    4: import java.util.ArrayList;
    5: import java.util.List;
    6: import java.util.concurrent.ExecutorService;
    7: import java.util.concurrent.Executors;
    8:
    9: import dev.gavinthomas.tictactoe.input.*;
   10: import dev.gavinthomas.tictactoe.opponents.Computer;
   11: import dev.gavinthomas.tictactoe.opponents.Player;
   12: import dev.gavinthomas.tictactoe.types.Opponent;
   13: import dev.gavinthomas.tictactoe.Board.PieceType;
   14: import dev.gavinthomas.tictactoe.Game;
   15: import dev.gavinthomas.tictactoe.types.UIHolder;
   16: import dev.gavinthomas.tictactoe.types.Visuals;
   17: import dev.gavinthomas.tictactoe.ui.Menu;
   18: import dev.gavinthomas.tictactoe.utils.Term;
   19: import dev.gavinthomas.tictactoe.utils.listeners;
   20:
   21: public class TicTacToe {
   22:   private List<Keybind> KBS = new ArrayList<>();
   23:   private List<Keybind> KBSDIS = new ArrayList<>();
   24:   private Game game;
   25:   private UIHolder currentUI;
   26:   private Keybind menuKBS;
   27:   public boolean invalidSize = true;
   28:   public final Term TERM = new Term();
   29:   private final Term privTerm = new Term();
   30:   public static TicTacToe CURR;
   31:   public final Point SIZE = new Point(0, 0);
   32:   public static final Point OFFSET = new Point(0, 0);
   33:
   34:   public TicTacToe() {
   35:     TicTacToe.CURR = this;
   36:     this.init();
   37:   }
   38:   public void init() {
   39:     new listeners.terminalResizeListener(this::resized);
   40:     input.init();
   41:     input.toggleRead(true);
   42:     TERM.requestSize(this::updateTermSize);
   43:     menuKBS = new Keybind(
   44:         new Keycode[] { Keycode.UP_ARROW, Keycode.DOWN_ARROW, Keycode.LEFT_ARROW
, Keycode.RIGHT_ARROW, Keycode.SPACE, Keycode.LOWER_L },
   45:         new Object[] { KeybindArgument.KEYCODE }, this::input);
   46:     input.addBinds(menuKBS);
   47: //    currentUI = new Menu();
   48: //    currentUI.render();
   49:   }
   50:
   51:   public void newGame(Game.Builder gameConfig) {
   52:     menuKBS.set.enabled(false);
   53:     this.game = new Game(gameConfig);
   54:     this.game.start();
   55:     menuKBS.set.enabled(true);
   56:   }
   57:
   58:   public void registerKB(Keybind kb) {
   59:     KBS.add(kb);
   60:     input.addBinds(kb);
   61:   }
   62:
```

```
   63:   public void registerKB(List<Keybind> kb) {
   64:     KBS.addAll(kb);
   65:     input.addBinds(kb);
   66:   }
   67:
   68:   public void deregisterKB(List<Keybind> kb) {
   69:     KBS.removeAll(kb);
   70:     input.removeBinds(kb);
   71:   }
   72:
   73:   public void resized() {
   74:     TERM.requestSize(this::updateTermSize);
   75:   }
   76:
   77:   public void render() {
   78:     if (game != null) {
   79:       game.render();
   80:       return;
   81:     }
   82:     if (currentUI == null) return;
   83:     currentUI.render();
   84:   }
   85:
   86:
   87:   public void renderSizeUI(int x, int y) {
   88:     TERM.clear(true);
   89:     TERM.hideCursor(true);
   90:     TERM.saveCursor();
   91:     System.out.print("â\226\210â\226\200â\226\200â\226\200\033[1B\033[4Dâ\226
\210");
   92:     TERM.setCursorPos(x - 3, 1);
   93:     System.out.print("â\226\200â\226\200â\226\200â\226\210\033[1Bâ\226\210");
   94:     TERM.setCursorPos(1, y - 1);
   95:     System.out.print("â\226\210\033[1B\033[2Dâ\226\210â\226\204â\226\204â\226
\204");
   96:     TERM.setCursorPos(x, y - 1);
   97:     System.out.print("â\226\210\033[1B\033[3Dâ\226\204â\226\204â\226\204â\226
\210");
   98:     TERM.restoreCursor();
   99:     if (x < 30 || y < 10) return;
  100:     TERM.setCursorPos((int) Math.ceil((x / 2.0) - 9.5), (y / 2) - 2);
  101:     System.out.print(Visuals.sizeUI(x, y, 100, 32));
  102: //     System.out.print(Visuals.sizeBorder(x, y));
  103:   }
  104:
  105:   public void updateTermSize(int x, int y) {
  106:     SIZE.x = x;
  107:     SIZE.y = y;
  108:     if (x < 100 || y < 32) {
  109:       if (currentUI instanceof Menu) ((Menu) currentUI).endTasks();
  110:       this.invalidSize = true;
  111:       for (Keybind kb : KBS) {
  112:         kb.set.forceDisabled(true);
  113:       }
  114:       this.renderSizeUI(x, y);
  115:       return;
  116:     }
  117:
  118:     this.invalidSize = false;
  119:     for (Keybind kb : KBS) {
  120:       kb.set.forceDisabled(false);
  121:     }
  122:     TicTacToe.OFFSET.x = (x - 100) / 2;
```

```
123:     TicTacToe.OFFSET.y = (y - 32) / 2;
124:     this.render();
125:   }
126:
127:   public void updateTermSize(Object[] args) {
128: //    System.out.println("args1");
129:     if (!(args[0] instanceof Point)) return;
130: //    System.out.println("args2");
131:     updateTermSize(((Point) args[0]).x, ((Point) args[0]).y);
132:   }
133:
134:   public void input(Object[] args) {
135:
136:   }
137: }
```

```
    1: package out;
    2:
    3: import java.io.FileWriter;
    4: import java.io.BufferedWriter;
    5: import java.io.IOException;
    6:
    7: public abstract class Out {
    8:   public static void append(String str) {
    9:     try {
   10:       BufferedWriter writer = new BufferedWriter(new FileWriter("output.txt", tr
ue));
   11:       writer.append(str);
   12:       writer.append('\n');
   13:
   14:       writer.close();
   15:     } catch (IOException e) {
   16:     }
   17:   }
   18:
   19:   public static void append(Object val) {
   20:     append(String.valueOf(val));
   21:   }
   22:   public static void append(int val) {
   23:     append(String.valueOf(val));
   24:   }
   25:   public static void append(byte val) {
   26:     append(String.valueOf(val));
   27:   }
   28:   public static void append(char val) {
   29:     append(String.valueOf(val));
   30:   }
   31:   public static void append(double val) {
   32:     append(String.valueOf(val));
   33:   }
   34:   public static void append(long val) {
   35:     append(String.valueOf(val));
   36:   }
   37:   public static void append(float val) {
   38:     append(String.valueOf(val));
   39:   }
   40:   public static void append(boolean val) {
   41:     append(String.valueOf(val));
   42:   }
   43: }
```