Hashi Puzzle (Islands and Bridges) Technical Document

Created By:

Wojciech Teodorowicz

Technical Document

Student Number: C00193622

Supervisor: Joseph Kehoe

Institute of Technology, Carlow

02nd September 2018

Contents

Code.

BoardElement.java

```
package Island_and_Bridges.Hashi;

import android.util.Log;

//this class holds information about the grid
public class BoardElement {
  // ******************************************************
  // ******************************************************
  // General members
  // ******************************************************
  // ******************************************************

  public int row = 0;
  public int col = 0;

  // Island?
  public boolean is_island = false;

//Island Memebers
  public int max_connecting_bridges = 0;
  public int column = 0;

  // It is easier to refer to neighbours via directions.
  public enum Direction {
    EAST, SOUTH, WEST, NORTH;
  }

  // Pairs of a BoardElement and the number of connecting bridges
  public Connection connecting_north = null;
  public Connection connecting_south = null;
  public Connection connecting_east = null;
  public Connection connecting_west = null;

  public BoardElement clone() {
    BoardElement elt = new BoardElement();
    elt.row = row;
    elt.col = col;
    Log.i("BoardElemnet", "Cloning " + "row " + elt.row + " " + "column " +
elt.col);

    elt.max_connecting_bridges = max_connecting_bridges;
    elt.is_island = is_island;
    if (connecting_east != null)
      elt.connecting_east = new Connection();
    else
      elt.connecting_east = null;

    if (connecting_north!= null)
      elt.connecting_north = new Connection();
    else
      elt.connecting_north = null;

    if (connecting_south!= null)
      elt.connecting_south = new Connection();
    else
      elt.connecting_south = null;

    if (connecting_west != null)
      elt.connecting_west = new Connection();
```

```
    else
      elt.connecting_west = null;

    return elt;
  }

  private int GetConnectionCount(Connection connection){
    if (connection == null) {
      return 0;
    } else {
      return 1;
    }
  }

  // Return the current count of connections connected
  // to this island.
  public int GetCurrentCount() {
    if (!is_island) {
      return 0;
    }
    int s = GetConnectionCount(connecting_east);
    s += GetConnectionCount(connecting_south);
    s += GetConnectionCount(connecting_north);
    s += GetConnectionCount(connecting_west);
    return s;
  }

  void AddConnection(Direction dir, BoardElement dest, int value) {
    Connection connection = null;
    switch (dir) {
      case EAST:
        connecting_east = new Connection();
        connection = connecting_east;
        break;
      case WEST:
        connecting_west = new Connection();
        connection = connecting_west;
        break;
      case SOUTH:
        connecting_south = new Connection();
        connection = connecting_south;
        break;
      case NORTH:
        connecting_north = new Connection();
        connection = connecting_north;
        break;
    }
  }
}; // BoardElement
```

MapView.java

```
package Island_and_Bridges.Hashi;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Rect;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.os.Vibrator;

import static junit.framework.Assert.*;
```

```
public class MapView extends View {
  private BoardCreation board_state;
  final private Paint background = new Paint();
  final private Paint cell_lines = new Paint();
  final private Paint text_paint = new Paint();
  final private Paint bridge_paint = new Paint();

  private PixelPosition[][] positions;

  private boolean drawing_bridge = false;
  private boolean live_bridge_legal = false;
  private PixelPosition start_candidate = null;
  private PixelPosition end_candidate = null;

  private float end_point_x = 0;
  private float end_point_y = 0;

  private final Bitmap default_icon =
BitmapFactory.decodeResource(this.getResources(),R.drawable.yellowicon);
  private final Bitmap selected_icon =
BitmapFactory.decodeResource(this.getResources(),R.drawable.greenicon);
  private final Bitmap done_icon =
BitmapFactory.decodeResource(this.getResources(),R.drawable.greenicon);


  class PixelPosition {
    public float x;
    public float y;
    public PixelPosition(float x, float y) {
      this.x = x;
      this.y = y;
    }
    public BoardElement element_reference = null;
    public boolean is_selected = false;
  }

  public MapView(Context context, BoardCreation state) {
    super(context);
    setFocusable(true);
    setFocusableInTouchMode(true);
    board_state = state;
    cell_lines.setColor(0xFFc0c0c0);
    text_paint.setColor(0xff00003f);
    text_paint.setTextSize(20);
    Paint text_paint_done = new Paint();
    text_paint_done.setColor(0xffff003f);
    text_paint_done.setTextSize(20);
    bridge_paint.setStrokeWidth(5);
    bridge_paint.setColor(0xff000000);
    Reset();
  }

  public void Reset() {
    int board_width = board_state.getCurrentState().board_width;
    positions  = new PixelPosition[board_width][board_width];
    for (int row = 0; row < board_width; ++row) {
      for (int col= 0; col < board_width; ++col) {
   positions[row][col] = new PixelPosition(0, 0);
   GetCellCenter(row, col,positions[row][col]);
   positions[row][col].element_reference =
          board_state.getCurrentState().board_elements[row][col];
      }
    }
  }

  @Override
  protected void onSizeChanged(int newx, int newy, int oldx, int oldy) {
    Reset();
```

```
  }

  @Override
  protected void onDraw(Canvas canvas) {
    final int board_width = board_state.getCurrentState().board_width;
    // Draw the background...
    background.setColor(getResources().getColor(
          R.color.background));
    canvas.drawRect(0, 0, getWidth(), getHeight(), background);

    // We draw a grid of the size of the board.
    for (int i = 0; i < board_width; ++i) {
      PixelPosition p0 = positions[i][0];
      PixelPosition p1 = positions[i][board_width - 1];
      canvas.drawLine(p0.x, p0.y, p1.x,p1.y, cell_lines);
      p0 = positions[0][i];
      p1 = positions[board_width - 1][i];
      canvas.drawLine(p0.x, p0.y, p1.x,p1.y, cell_lines);
    }

    // Draw accepted bridges.
    // From top to bottom and left to right.
    // We exploit symmetry.
    for (int row = 0; row < board_width; ++row) {
      for (int col= 0; col < board_width; ++col) {
        final String logmarker = getClass().getName() + " " + row + " " + col;
        BoardElement elt = null;
        elt = board_state.getCurrentState().board_elements[row][col];
        assertNotNull(logmarker + ": null element", elt);
        if (elt.is_island) {
          if (elt.connecting_east != null) {
            assertNotNull(logmarker + ": connecting east destination is null",
elt.connecting_east.destination);
            PaintBridge(canvas, elt,
elt.connecting_east.destination,elt.connecting_east.second);
          }
          if (elt.connecting_south!= null) {
            assertNotNull(logmarker + ": connecting south destination is null",
elt.connecting_south.destination);
            PaintBridge(canvas, elt,
elt.connecting_south.destination,elt.connecting_south.second);
          }
        }
      }
    }

    // We draw live bridge lines if needed.
    if (drawing_bridge) {
      canvas.drawLine(start_candidate.x, start_candidate.y,
          end_point_x, end_point_y, bridge_paint);
    }

    // On top of this we draw possibly two sets of icons.
    // The not selected numbers and the selected ones.
    for (int row = 0; row < board_width; ++row) {
      for (int col= 0; col < board_width; ++col) {
        if (board_state.getCurrentState().board_elements[row][col].is_island) {
          PixelPosition p = positions[row][col];
          Bitmap icon = null;
          if (p.is_selected) {
            icon = selected_icon;
          } else {
            icon = default_icon;
          }
          int max_connecting_bridges =

board_state.getCurrentState().board_elements[row][col].max_connecting_bridges;
          int count =
```

```
board_state.getCurrentState().board_elements[row][col].GetCurrentCount();
          PaintNumber(canvas, p, max_connecting_bridges, count ==
max_connecting_bridges, icon);
        }
      }
    }

  }

  private Rect paint_source = new Rect();
  private Rect paint_destination = new Rect();

  private void PaintNumber(Canvas canvas, PixelPosition p, int number, boolean
done, Bitmap icon) {
    float x0 = p.x;
    float y0 = p.y;
    float x = p.x - text_paint.getTextSize() / 2 + 4;
    float y = p.y + text_paint.getTextSize() / 2 - 2;

    paint_source.left = 0;
    paint_source.right = icon.getWidth();
    paint_source.top = 0;
    paint_source.bottom = icon.getHeight();
    paint_destination.left = (int) (x0 - 12);
    paint_destination.top = (int) (y0 - 12);
    paint_destination.right= (int) (x0 + 12);
    paint_destination.bottom= (int) (y0 + 12);
    if (!done)
      canvas.drawBitmap(icon, paint_source, paint_destination, text_paint);
    else
      canvas.drawBitmap(done_icon, paint_source, paint_destination, text_paint);
    canvas.drawText(String.format("%d", number), x, y, text_paint);
  }

  private void PaintBridge(Canvas canvas, BoardElement start, BoardElement end, int
bridges) {
    float startX = positions[start.row][start.col].x;
    float startY = positions[start.row][start.col].y;

    float endX = positions[end.row][end.col].x;
    float endY = positions[end.row][end.col].y;
    float linewidth = bridge_paint.getStrokeWidth();
    if (bridges == 1) {
      canvas.drawLine(startX, startY, endX, endY, bridge_paint);
    } else if (bridges == 2) {
      if (startY == endY) {
   canvas.drawLine(startX, startY - linewidth - 2, endX, endY - linewidth -2,
bridge_paint);
   canvas.drawLine(startX, startY + linewidth + 2, endX, endY + linewidth + 2,
bridge_paint);
      } else {
   canvas.drawLine(startX - linewidth - 2, startY, endX - linewidth - 2, endY,
bridge_paint);
   canvas.drawLine(startX + linewidth + 2, startY, endX + linewidth +2, endY,
bridge_paint);
       bridges = 1;
      }
    }
  }

  @Override
  public boolean onTouchEvent(MotionEvent event) {
    if (event.getAction() == MotionEvent.ACTION_DOWN ||
        event.getAction() == MotionEvent.ACTION_MOVE ||
        event.getAction() == MotionEvent.ACTION_UP)  {
      float x =  event.getX();
      float y = event.getY();
```

```
      end_point_x = x;
      end_point_y = y;
      PixelPosition candidate = null;
      switch(event.getAction()) {
        case MotionEvent.ACTION_DOWN:
          Log.d(getClass().getName(), "ACTION DOWN");
          candidate =  TrySelect(x, y);
          if (candidate != null) {
            drawing_bridge = true;
            candidate.is_selected = true;
            start_candidate = candidate;
            Vibrator vibrator = (Vibrator)
getContext().getSystemService(Context.VIBRATOR_SERVICE);
            vibrator.vibrate(50L);
            return true;
          }

          start_candidate = null;

          break;
        case MotionEvent.ACTION_MOVE:
          if (drawing_bridge) {
            float dx = start_candidate.x - x;
            float dy = start_candidate.y - y;
            candidate = TrySelect(x,y);
            if (candidate != null && candidate != start_candidate) {
              // Allow only non-diagonal neighbors.
              if (candidate.x == start_candidate.x ||
                  candidate.y == start_candidate.y) {
                candidate.is_selected = true;
              }
              if (candidate != end_candidate && end_candidate != null) {
                end_candidate.is_selected = false;
              }
              if (end_candidate != candidate) {
                Vibrator vibrator = (Vibrator)
getContext().getSystemService(Context.VIBRATOR_SERVICE);
                vibrator.vibrate(50L);
              }
              end_candidate = candidate;
            } else {
              if (end_candidate != null) {
                end_candidate.is_selected = false;
              }

              end_candidate = null;
            }
          }
          break;
        case MotionEvent.ACTION_UP:
          Log.d(getClass().getName(), "ACTION UP");

          if (start_candidate == null)
            return false;

          drawing_bridge = false;
          candidate = TrySelect(x, y);
          boolean result = false;
          if (candidate != null) {
            BoardElement start_elt = start_candidate.element_reference;
            BoardElement end_elt = candidate.element_reference;
            result = board_state.TryAddNewBridge(start_elt, end_elt, 1);
          }

          // Cleanup
          start_candidate.is_selected = false;
          start_candidate = null;
          if (end_candidate != null) {
```

```
            end_candidate.is_selected = false;
            end_candidate = null;
          }

          if (result) {
            CheckEndCondition();
          }
        }
        invalidate();
        return true;
      }
      return super.onTouchEvent(event);
    }

    void CheckEndCondition() {
    }

    private PixelPosition TrySelect(float x, float y) {
      for (PixelPosition[] position : positions) {
        for (int j = 0; j < position.length; ++j) {
          double sq_distance = Math.sqrt((position[j].x - x) * (position[j].x - x) +
                  (position[j].y - y) * (position[j].y - y));
          float MAX_DISTANCE_FOR_SELECTION = 20;
          if (sq_distance < MAX_DISTANCE_FOR_SELECTION &&
                  position[j].element_reference != null &&
                  position[j].element_reference.is_island) {
            Log.d(getClass().getName(),
                    String.format("Choosing %d %d",
                            position[j].element_reference.row,
                            position[j].element_reference.col));
            return position[j];
          }
        }
      }
      return null;
    }

    private void GetCellCenter(int board_row, int board_column, PixelPosition result)
{
      final int board_width = board_state.getCurrentState().board_width;
      final int grid_start = 5;
      final int cell_width = (getWidth() - 2 * grid_start) / board_width;
      final int cell_height = (getHeight() - 2 * grid_start) / board_width;
      float x = grid_start + board_column * cell_width + cell_width / 2;
      float y = grid_start + board_row * cell_height  + cell_height / 2;
      result.x = x;
      result.y = y;
    }

}
```

Hashi_Main.java

```
package Island_and_Bridges.Hashi;


import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;

/** Called when the activity is first created. */
public class Hashi_Main extends Activity implements OnClickListener {

  @Override
    public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Set up click listeners for all the buttons
        View continueButton = findViewById(R.id.continue_button);
        continueButton.setOnClickListener(this);

        View newButton = findViewById(R.id.new_button);
        newButton.setOnClickListener(this);

        View aboutButton = findViewById(R.id.about_button);
        aboutButton.setOnClickListener(this);

        View exitButton = findViewById(R.id.exit_button);
        exitButton.setOnClickListener(this);

    }

  // click handling
  public void onClick(View view) {
    switch (view.getId()) {
      case R.id.exit_button:
   finish();
   break;
      case R.id.new_button:
   NewGame();
   break;
    }
  }


  // Create a new game.
  public void NewGame() {
    // We first ask for the difficulty level.
    new AlertDialog.Builder(this)
      .setTitle(R.string.new_game_title)
      // we provide a char array with the on click listener.
      .setItems(R.array.difficulty,
       new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialoginterface, int hardness) {
          Intent intent = new Intent(Hashi_Main.this, HashiGame.class);
          intent.putExtra(HashiGame.KEY_DIFFICULTY, hardness);
          startActivity(intent);
        }
       })
    .show();
  }
}
```

Connection.java

```
package Island_and_Bridges.Hashi;

/**
 * Created by Wojciech on 19/03/2018.
 */

import android.util.Log;

import org.w3c.dom.Node;

public class Connection {
    int distance = 0;
    int number = 0;

    Node node;
    public BoardElement destination;
    public int second = 0;
```

```
    public void setNode (Node node) {
        this.node = node;
    }

    public Node getNode () {
        return node;
    }

    public int getDistance() {
        return distance;
    }
    public void setDistance(int distance) {
        this.distance = distance;
    }
    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }

    public void increment () {
        if (node != null)
            Log.d("Connections", "updating non existent node");
        setNumber (getNumber() + 1);
    }
}
```

BoardCreation.java

```
package Island_and_Bridges.Hashi;

import android.util.Log;


  public class BoardElement {
    // ********************************************************
    // ********************************************************
    // General members
    // ********************************************************
    // ********************************************************

    public int row = 0;
    public int col = 0;

    // Is this an island?
    public boolean is_island = false;

    // ********************************************************
    // ********************************************************
    // Island specific members.
    // ********************************************************
    // ********************************************************
    public int max_connecting_bridges = 0;

    // It is easier to refer to neighbours via directions.
    public enum Direction {
      EAST, SOUTH, WEST, NORTH;
    }

    // Pairs of a BoardElement and the number of connecting bridges
    public Connection connecting_north = null;
    public Connection connecting_south = null;
    public Connection connecting_east = null;
    public Connection connecting_west = null;

    public BoardElement clone() {
      BoardElement elt = new BoardElement();
```

```
      elt.row = row;
      elt.col = col;
      Log.i("BoardElemnet", "Cloning" + elt.row + " " + elt.col);

      elt.max_connecting_bridges = max_connecting_bridges;
      elt.is_island = is_island;
      if (connecting_east != null)
        elt.connecting_east = new Connection();
      else
        elt.connecting_east = null;

      if (connecting_north!= null)
        elt.connecting_north = new Connection();
      else
        elt.connecting_north = null;

      if (connecting_south!= null)
        elt.connecting_south = new Connection();
      else
        elt.connecting_south = null;

      if (connecting_west != null)
        elt.connecting_west = new Connection();
      else
        elt.connecting_west = null;

      return elt;
    }

    private int GetConnectionCount(Connection connection){
      if (connection == null) {
        return 0;
      } else {
        return connection.second;
      }
    }

    // Return the current count of connections connected
    // to this island.
    public int GetCurrentCount() {
      if (!is_island) {
        return 0;
      }
      int s = GetConnectionCount(connecting_east);
      s += GetConnectionCount(connecting_south);
      s += GetConnectionCount(connecting_north);
      s += GetConnectionCount(connecting_west);
      return s;
    }

    void AddConnection(Direction dir, BoardElement dest, int value) {
      Connection connection = null;
      switch (dir) {
        case EAST:
          connecting_east = new Connection(dest, value);
          connection = connecting_east;
          break;
        case WEST:
          connecting_west = new Connection(dest, value);
          connection = connecting_west;
          break;
        case SOUTH:
          connecting_south = new Connection(dest, value);
          connection = connecting_south;
          break;
        case NORTH:
          connecting_north = new Connection(dest, value);
          connection = connecting_north;
```

```
            break;
        }
      }
    }; // BoardElement
```

HashiGame.java

```
/**
 *
 */
package Island_and_Bridges.Hashi;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;

import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.client.AWSStartupHandler;
import com.amazonaws.mobile.client.AWSStartupResult;

import java.text.DecimalFormat;
import java.text.FieldPosition;
import java.text.NumberFormat;
import java.util.Locale;
import java.util.TimerTask;

/**
 * @author ghb
 *
 */
public class HashiGame extends Activity implements OnClickListener {


  public static final String KEY_DIFFICULTY =
    "de.Hashi.difficulty";

  // Commands
  public static final int RESTART = 1;
  public static final int EXIT = 2;
  public static final int TIME = 3;
  public static final int DIFFICULTY_EASY = 0;
  public static final int DIFFICULTY_MEDIUM = 1;
  public static final int DIFFICULTY_HARD = 2;

  private BoardCreation boardstate;
  private MapView view;

  private void ResetGameState(boolean reset_timer) {
    int diff = getIntent().getIntExtra(KEY_DIFFICULTY, DIFFICULTY_EASY);
    BoardCreation oldstate = (BoardCreation) getLastNonConfigurationInstance();
    if (boardstate == null && oldstate == null) {
      Log.d(getClass().getName(), "no old boardstate ");
      boardstate = new BoardCreation(diff);
    }
    if (boardstate == null && oldstate != null) {
      boardstate = oldstate;
    }
    if (boardstate != null && oldstate == null) {
      boardstate.ResetGame();
    }
```

```
    if (reset_timer) {

      gametimer.purge();
    }
  }

  @Override
  public Object onRetainNonConfigurationInstance() {
      return boardstate;
  }

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
      AWSMobileClient.getInstance().initialize(this, new AWSStartupHandler() {
          @Override
          public void onComplete(AWSStartupResult awsStartupResult) {
              Log.d("YourMainActivity", "AWSMobileClient is instantiated and you
are connected to AWS!");
          }
      }).execute();
    ResetGameState(true);

    view = new MapView(this, boardstate);
    LinearLayout l = new LinearLayout(this);
    l.setOrientation(LinearLayout.VERTICAL);

    // Create a LinearLayout in which to add the ImageView
    LinearLayout lheader = new LinearLayout(this);
    lheader.setOrientation(LinearLayout.HORIZONTAL);

    TextView time = new TextView(this);
    time.setId(TIME);
    lheader.addView(time);

    Button b = new Button(this);
    b.setText("Restart");
    b.setId(RESTART);
    b.setOnClickListener(this);
    lheader.addView(b);

    b = new Button(this);
    b.setText("Exit");
    b.setId(EXIT);
    b.setOnClickListener(this);
    lheader.addView(b);
    l.addView(lheader);

    l.addView(view);
    // Add the ImageView to the layout and set the layout as the content view
    setContentView(l);

    gametimer.schedule(new TimerTask() {
      public void run() {
        seconds += 0.1;
        hRefresh.sendEmptyMessage(0);
      }
    }, 100, 100);

    view.requestFocus();
  }

  Handler hRefresh = new Handler(){
    @Override
      public void handleMessage(Message msg) {
   updatetime();
      }
  };
```

```
  float seconds = 0;

  java.util.Timer gametimer = new java.util.Timer();

  public static String format(double x, int width, int precision) {
    StringBuffer strBuf = new StringBuffer();
    String str = new String();
    DecimalFormat decForm =
      (DecimalFormat) NumberFormat.getInstance(Locale.ITALIAN);
    FieldPosition intPos = new FieldPosition(DecimalFormat.INTEGER_FIELD);
    decForm.setMinimumFractionDigits(precision);
    decForm.setMaximumFractionDigits(precision);
    decForm.format(x, strBuf, intPos);
    for (int i = 0; i < width - strBuf.length(); i++) {
      str += " ";
    }
    return (str + strBuf);
  }

  void updatetime() {
    TextView t = (TextView)findViewById(TIME);
    t.setText("Elapsed Time: "+ format(seconds, 5, 1 ));
    t.postInvalidate();
  }


  @Override
    public void onClick(View view) {
      switch (view.getId()) {
   case RESTART:
     ResetGameState(true);
     this.view.Reset();
     this.view.invalidate();
     break;
   case EXIT:
     ResetGameState(true);
     finish();
     break;
      }
    }

  @Override
    protected void onResume() {
      // Load sate
      super.onResume();
    }

  @Override
    protected void onPause() {
      // Store sate
      super.onPause();
    }

}
```

Land.java

```
package Island_and_Bridges.Hashi;
import  Island_and_Bridges.Hashi.BoardElement;
import android.annotation.TargetApi;
import android.graphics.Path;
import android.os.Build;
import android.util.Log;

import java.util.Arrays;

import static Island_and_Bridges.Hashi.BoardElement.*;
import static Island_and_Bridges.Hashi.BoardElement.Direction.EAST;
import static Island_and_Bridges.Hashi.BoardElement.Direction.NORTH;
import static Island_and_Bridges.Hashi.BoardElement.Direction.SOUTH;
import static android.graphics.Path.Direction.*;
// This is a DFS algorithm
// this algorithm was designed to recursively searchj through the array
// and connect bridges so that the map is solvable


public class Land {

    private int[][] BRIDGES_TO_BUILD;

    private boolean[][] IS_ISLAND;
    private Direction[][] BRIDGES_ALREADY_BUILT;

    public Land(int[][] bridgesToDo){
        BRIDGES_TO_BUILD = copy(bridgesToDo);

        int numberRows = bridgesToDo.length;
        int numberColumns = bridgesToDo[0].length;
        BRIDGES_ALREADY_BUILT = new Direction[numberRows][numberColumns];
        IS_ISLAND = new boolean[numberRows][numberColumns];
        for(int i=0;i<numberRows;i++) {
            for (int j = 0; j < numberColumns; j++) {
                BRIDGES_ALREADY_BUILT[i][j] = null;
                IS_ISLAND[i][j] = bridgesToDo[i][j] > 0;
            }
        }
    }

    public Land(Land other){
        BRIDGES_TO_BUILD = copy(other.BRIDGES_TO_BUILD);
        int numberRows = BRIDGES_TO_BUILD.length;
        int numberColumns =  BRIDGES_TO_BUILD[0].length;
        BRIDGES_ALREADY_BUILT = new Direction[numberRows][numberColumns];
        IS_ISLAND = new boolean[numberRows][numberColumns];
        for(int i=0;i<numberRows;i++) {
            for (int j = 0; j < numberColumns; j++) {
                BRIDGES_ALREADY_BUILT[i][j] = other.BRIDGES_ALREADY_BUILT[i][j];
                IS_ISLAND[i][j] = other.IS_ISLAND[i][j];
            }
        }
    }

    public int[] next(int row, int column, Direction dir){
        int numberRows = BRIDGES_TO_BUILD.length;
        int numberColumns = BRIDGES_TO_BUILD[0].length;

        // out of bounds
        if(row < 0 || row >=numberRows || column < 0 || column >= numberColumns)
            return null;


        // motion vectors
        int[][] motionVector = {{-1, 0},{0,1},{1,0},{0,-1}};
```

```
        int i = Arrays.asList(Direction.values()).indexOf(dir);

        // calculate next
        int[] out = new int[]{row + motionVector[i][0], column +
motionVector[i][1]};

        row = out[0];
        column = out[1];

        // out of bounds
        if(row < 0 || row >=numberRows || column < 0 || column >= numberColumns)
            return null;

        // return
        return out;
    }

    public int[] nextIsland(int row, int column, Direction dir){
        int[] tmp = next(row,column,dir);
        if(tmp == null)
            return null;
        while(!IS_ISLAND[tmp[0]][tmp[1]]){
            tmp = next(tmp[0], tmp[1], dir);
            if(tmp == null)
                return null;
        }
        return tmp;
    }

    public boolean canBuildBridge(int row0, int column0, int row1, int column1){
        if(row0 == row1 && column0 > column1){
            return canBuildBridge(row0, column1, row1, column0);
        }
        if(column0 == column1 && row0 > row1){
            return canBuildBridge(row1, column0, row0, column1);
        }
        if(row0 == row1){
            int[] tmp = nextIsland(row0, column0, Direction.EAST);
            if(tmp == null)
                return false;
            if(tmp[0] == row1 || tmp[1] == column1)
                return false;
            if(BRIDGES_TO_BUILD[row0][column0] == 0)
                return false;
            if(BRIDGES_TO_BUILD[row1][column1] == 0)
                return false;
            for (int i = column0; i <= column1 ; i++) {
                if(IS_ISLAND[row0][i])
                    continue;
                if(BRIDGES_ALREADY_BUILT[row0][i] == Direction.NORTH)
                    return false;
            }
        }
        if(column0 == column1){
            int[] tmp = nextIsland(row0, column0, Direction.SOUTH);
            if(tmp == null)
                return false;
            if(tmp[0] == row1 || tmp[1] == column1)
                return false;
            if(BRIDGES_TO_BUILD[row0][column0] == 0 ||
BRIDGES_TO_BUILD[row1][column1] == 0)
                return false;
            for (int i = row0; i <= row1 ; i++) {
                if(IS_ISLAND[i][column0])
                    continue;
                if(BRIDGES_ALREADY_BUILT[i][column0] == Direction.EAST)
                    return false;
            }
```

```
        }
        // default
        return true;
    }

    public int[] lowestTodo(){
        int R = BRIDGES_TO_BUILD.length;
        int C = BRIDGES_TO_BUILD[0].length;

        int[] out = {0, 0};
        for (int i=0;i<R;i++) {
            for (int j = 0; j < C; j++) {
                if(BRIDGES_TO_BUILD[i][j] == 0)
                    continue;
                if (BRIDGES_TO_BUILD[out[0]][out[1]] == 0)
                    out = new int[]{i, j};
                if (BRIDGES_TO_BUILD[i][j] < BRIDGES_TO_BUILD[out[0]][out[1]])
                    out = new int[]{i, j};
            }
        }
        if (BRIDGES_TO_BUILD[out[0]][out[1]] == 0) {
            return null;
        }
        return out;
    }

    @TargetApi(Build.VERSION_CODES.GINGERBREAD)
    private int[][] copy(int[][] other){
        int[][] out = new int[other.length][other.length == 0 ? 0 :
other[0].length];
        for(int r=0;r<other.length;r++)
            out[r] = Arrays.copyOf(other[r], other[r].length);
        return out;
    }

    public void connect(int row0, int column0, int row1, int column1){
        if(row0 == row1 && column0 > column1){
            connect(row0, column1, row1, column0);
            return;
        }
        if(column0 == column1 && row0 > row1){
            connect(row1, column0, row0, column1);
            return;
        }
        if(!canBuildBridge(row0, column0, row1, column1))
            return;

        BRIDGES_TO_BUILD[row0][column0]--;
        BRIDGES_TO_BUILD[row1][column1]--;

        if(row0 == row1){
            for (int i = column0; i <= column1 ; i++) {
                if(IS_ISLAND[row0][i])
                    continue;
                BRIDGES_ALREADY_BUILT[row0][i] = Direction.EAST;
            }
        }
        if(column0 == column1){
            for (int i = row0; i <= row1 ; i++) {
                if(IS_ISLAND[i][column0])
                    continue;
                BRIDGES_ALREADY_BUILT[i][column0] = Direction.NORTH;
            }
        }
    }
}
```

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
android:id="@+id/widget30"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="@color/lightgray"
android:padding="5px"
android:orientation="vertical"
android:gravity="center"
xmlns:android="http://schemas.android.com/apk/res/android"
>
<Button
android:id="@+id/continue_button"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/continue_label"
>
</Button>
<Button
android:id="@+id/new_button"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/new_game_label"
>
</Button>
<Button
android:id="@+id/about_button"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/about_label"
>
</Button>
<Button
android:id="@+id/exit_button"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/exit_label"
>
</Button>
</LinearLayout>
```

arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <array name="difficulty">
      <item>@string/easy_label</item>
      <item>@string/medium_label</item>
      <item>@string/hard_label</item>
   </array>
</resources>
```

color.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
```

```
<color name="lightgray">#FFc0c0c0</color>
<color name="background">#00C322</color>
</resources>
```

strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Hashi0</string>
    <string name="game_title">Hashi0</string>
<!-- Main    -->
   <string name="continue_label">Continue</string>
   <string name="new_game_label">New Game</string>
   <string name="about_label">About</string>
   <string name="exit_label">Exit</string>

   <string name="new_game_title">Difficulty</string>
   <string name="easy_label">Easy</string>
   <string name="medium_label">Medium</string>
   <string name="hard_label">Hard</string>

<!--    -->

</resources>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="Island_and_Bridges.Hashi"
      android:versionCode="1"
      android:versionName="1.0">
  <uses-permission android:name="android.permission.VIBRATE"/>
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name="Island_and_Bridges.Hashi.Hashi_Main"
                  android:label="@string/app_name"
        android:screenOrientation="portrait" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
   <activity
         android:name="Island_and_Bridges.Hashi.HashiGame"
         android:label="@string/game_title"
         android:configChanges="keyboardHidden|orientation"
         android:screenOrientation="portrait" >
   </activity>
    </application>
</manifest>
```

Build.gradle

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 26
    buildToolsVersion "27.0.2"

    defaultConfig {
        minSdkVersion 26
        applicationId "Island_and_Bridges.Hashi"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.txt'
        }
```

```
    }
}
dependencies {
    implementation ('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar') {
transitive = true }
}
```