```
/**
 * Blockly Games: Robotmaze
 *
 * Copyright 2012 Google Inc.
 * https://github.com/google/blockly-games
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

/**
 * @fileoverview JavaScript for Blockly's Robotmaze application.
 * @author fraser@google.com (Neil Fraser)
 */
'use strict';

goog.provide('Robotmaze');

goog.require('Blockly.FieldDropdown');
goog.require('BlocklyDialogs');
goog.require('BlocklyGames');
goog.require('BlocklyInterface');
goog.require('Maze.Blocks');
goog.require('Robotmaze.soy');

BlocklyGames.NAME = 'robotmaze';

/**
 * Go to the next level.  Add skin parameter.
 * @suppress {duplicate}
 */
BlocklyInterface.nextLevel = function() {
  if (BlocklyGames.LEVEL < BlocklyGames.MAX_LEVEL) {
    window.location = window.location.protocol + '//' +
        window.location.host + window.location.pathname +
        '?lang=' + BlocklyGames.LANG + '&level=' + (BlocklyGames.LEVEL + 1) +
        '&skin=' + Robotmaze.SKIN_ID;
  } else {
    BlocklyInterface.indexPage();
  }
};

Robotmaze.MAX_BLOCKS = [undefined, // Level 0.
    Infinity, Infinity, 2, 5, 5, 5, 5, 10, 7, 10][BlocklyGames.LEVEL];

// Crash type constants.
Robotmaze.CRASH_STOP = 1;
Robotmaze.CRASH_SPIN = 2;
Robotmaze.CRASH_FALL = 3;

Robotmaze.SKINS = [
  // sprite: A 1029x51 set of 21 avatar images.
  // tiles: A 250x200 set of 20 map images.
  // marker: A 20x34 goal image.
  // background: An optional 400x450 background image, or false.
  // graph: Colour of optional grid lines, or false.
  // look: Colour of sonar-like look icon.
  // winSound: List of sounds (in various formats) to play when the player wins.
  // crashSound: List of sounds (in various formats) for player crashes.
  // crashType: Behaviour when player crashes (stop, spin, or fall).
  {
    sprite: 'robotmaze/pegman.png',
    tiles: 'robotmaze/tiles_pegman.png',
    marker: 'robotmaze/marker.png',
    background: false,
    graph: false,
    look: '#000',
    winSound: ['robotmaze/win.mp3', 'robotmaze/win.ogg'],
    crashSound: ['robotmaze/fail_pegman.mp3', 'robotmaze/fail_pegman.ogg'],
    crashType: Robotmaze.CRASH_STOP
  },
  {
    sprite: 'robotmaze/astro.png',
    tiles: 'robotmaze/tiles_astro.png',
    marker: 'robotmaze/marker.png',
    background: 'robotmaze/bg_astro.jpg',
    // Coma star cluster, photo by George Hatfield, used with permission.
    graph: false,
    look: '#fff',
    winSound: ['robotmaze/win.mp3', 'robotmaze/win.ogg'],
    crashSound: ['robotmaze/fail_astro.mp3', 'robotmaze/fail_astro.ogg'],
    crashType: Robotmaze.CRASH_SPIN
  },
  {
    sprite: 'robotmaze/panda.png',
    tiles: 'robotmaze/tiles_panda.png',
    marker: 'robotmaze/marker.png',
    background: 'robotmaze/bg_panda.jpg',
    // Spring canopy, photo by Rupert Fleetingly, CC licensed for reuse.
    graph: false,
    look: '#000',
    winSound: ['robotmaze/win.mp3', 'robotmaze/win.ogg'],
    crashSound: ['robotmaze/fail_panda.mp3', 'robotmaze/fail_panda.ogg'],
    crashType: Robotmaze.CRASH_FALL
  }
];
Robotmaze.SKIN_ID = BlocklyGames.getNumberParamFromUrl('skin', 0,
Robotmaze.SKINS.length);
Robotmaze.SKIN = Robotmaze.SKINS[Robotmaze.SKIN_ID];

/**
 * Milliseconds between each animation frame.
 */
Robotmaze.stepSpeed;

/**
 * The types of squares in the robotmaze, which is represented
 * as a 2D array of SquareType values.
 * @enum {number}
 */
Robotmaze.SquareType = {
  WALL: 0,
  OPEN: 1,
  START: 2,
  FINISH: 3
};

// The robotmaze square constants defined above are inlined here
// for ease of reading and writing the static robotmazes.
Robotmaze.map = [
// Level 0.
 undefined,
// Level 1.
 [[0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0],
  [0, 0, 2, 1, 3, 0, 0],
  [0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0]],
// Level 2.
 [[0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 1, 3, 0, 0, 0],
  [0, 0, 2, 1, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0]],
// Level 3.
 [[0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0],
  [0, 2, 1, 1, 1, 1, 3, 0],
  [0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0]],
// Level 4.
/**
 * Note, the path continues past the start and the goal in both directions.
 * This is intentionally done so users see the robotmaze is about getting from
 * the start to the goal and not necessarily about moving over every part of
 * the robotmaze, 'mowing the lawn' as Neil calls it.
 */
 [[0, 0, 0, 0, 0, 0, 0, 1],
  [0, 0, 0, 0, 0, 0, 1, 1],
  [0, 0, 0, 0, 0, 3, 1, 0],
  [0, 0, 0, 0, 1, 1, 0, 0],
  [0, 0, 0, 1, 1, 0, 0, 0],
  [0, 0, 1, 1, 0, 0, 0, 0],
  [0, 2, 1, 0, 0, 0, 0, 0],
  [1, 1, 0, 0, 0, 0, 0, 0]],
// Level 5.
 [[0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 3, 0, 0],
  [0, 0, 0, 0, 0, 1, 0, 0],
  [0, 0, 0, 0, 0, 1, 0, 0],
  [0, 0, 0, 0, 0, 1, 0, 0],
  [0, 0, 0, 0, 0, 1, 0, 0],
  [0, 0, 0, 2, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0]],
// Level 6.
 [[0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 1, 1, 1, 0, 0],
  [0, 1, 0, 0, 0, 1, 0, 0],
  [0, 1, 1, 3, 0, 1, 0, 0],
  [0, 0, 0, 0, 0, 1, 0, 0],
  [0, 2, 1, 1, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0]],
// Level 7.
 [[0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 1, 1, 0],
  [0, 2, 1, 1, 1, 1, 0, 0],
  [0, 0, 0, 0, 0, 1, 1, 0],
  [0, 1, 1, 3, 0, 1, 0, 0],
  [0, 1, 0, 1, 0, 1, 0, 0],
  [0, 1, 1, 1, 1, 1, 1, 0],
  [0, 0, 0, 0, 0, 0, 0, 0]],
// Level 8.
 [[0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 1, 1, 0, 0, 0],
  [0, 1, 0, 0, 1, 1, 0, 0],
  [0, 1, 1, 1, 0, 1, 0, 0],
  [0, 0, 0, 1, 0, 1, 0, 0],
  [0, 2, 1, 1, 0, 3, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0]],
// Level 9.
 [[0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 1, 1, 1, 0, 0],
  [0, 0, 1, 0, 0, 0, 0, 0],
  [3, 1, 1, 1, 1, 1, 1, 0],
  [0, 1, 0, 1, 0, 1, 1, 0],
  [1, 1, 1, 1, 1, 0, 1, 0],
  [0, 1, 0, 1, 0, 2, 1, 0],
  [0, 0, 0, 0, 0, 0, 0, 0]],
// Level 10.
 [[0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 0, 3, 0, 1, 0],
  [0, 1, 1, 0, 1, 1, 1, 0],
  [0, 1, 0, 1, 0, 1, 0, 0],
  [0, 1, 1, 1, 1, 1, 1, 0],
  [0, 0, 0, 1, 0, 0, 1, 0],
  [0, 2, 1, 1, 1, 0, 1, 0],
  [0, 0, 0, 0, 0, 0, 0, 0]]
][BlocklyGames.LEVEL];

/**
 * Measure robotmaze dimensions and set sizes.
 * ROWS: Number of tiles down.
 * COLS: Number of tiles across.
 * SQUARE_SIZE: Pixel height and width of each robotmaze square (i.e. tile).
 */
Robotmaze.ROWS = Robotmaze.map.length;
Robotmaze.COLS = Robotmaze.map[0].length;
Robotmaze.SQUARE_SIZE = 50;
Robotmaze.PEGMAN_HEIGHT = 52;
Robotmaze.PEGMAN_WIDTH = 49;

Robotmaze.MAZE_WIDTH = Robotmaze.SQUARE_SIZE * Robotmaze.COLS;
Robotmaze.MAZE_HEIGHT = Robotmaze.SQUARE_SIZE * Robotmaze.ROWS;
Robotmaze.PATH_WIDTH = Robotmaze.SQUARE_SIZE / 3;

/**
 * Constants for cardinal directions.  Subsequent code assumes these are
 * in the range 0..3 and that opposites have an absolute difference of 2.
 * @enum {number}
 */
Robotmaze.DirectionType = {
  NORTH: 0,
  EAST: 1,
  SOUTH: 2,
  WEST: 3
};

/**
 * Outcomes of running the user program.
 */
Robotmaze.ResultType = {
  UNSET: 0,
  SUCCESS: 1,
  FAILURE: -1,
  TIMEOUT: 2,
  ERROR: -2
};

/**
 * Result of last execution.
 */
Robotmaze.result = Robotmaze.ResultType.UNSET;

/**
 * Starting direction.
 */
Robotmaze.startDirection = Robotmaze.DirectionType.EAST;
```

```javascript

/**
 * PIDs of animation tasks currently executing.
 */
Robotmaze.pidList = [];

// Map each possible shape to a sprite.
// Input: Binary string representing Centre/North/West/South/East squares.
// Output: [x, y] coordinates of each tile's sprite in tiles.png.
Robotmaze.tile_SHAPES = {
  '10010': [4, 0],  // Dead ends
  '10001': [3, 3],
  '11000': [0, 1],
  '10100': [0, 2],
  '11010': [4, 1],  // Vertical
  '10101': [3, 2],  // Horizontal
  '10110': [0, 0],  // Elbows
  '10011': [2, 0],
  '11001': [4, 2],
  '11100': [2, 3],
  '11110': [1, 1],  // Junctions
  '10111': [1, 0],
  '11011': [2, 1],
  '11101': [1, 2],
  '11111': [2, 2],  // Cross
  'null0': [4, 3],  // Empty
  'null1': [3, 0],
  'null2': [3, 1],
  'null3': [0, 3],
  'null4': [1, 3]
};

/**
 * Create and layout all the nodes for the path, scenery, Pegman, and goal.
 */
Robotmaze.drawMap = function() {
  var svg = document.getElementById('svgRobotmaze');
  var scale = Math.max(Robotmaze.ROWS, Robotmaze.COLS) * Robotmaze.SQUARE_SIZE;
  svg.setAttribute('viewBox', '0 0 ' + scale + ' ' + scale);

  // Draw the outer square.
  var square = document.createElementNS(Blockly.SVG_NS, 'rect');
  square.setAttribute('width', Robotmaze.MAZE_WIDTH);
  square.setAttribute('height', Robotmaze.MAZE_HEIGHT);
  square.setAttribute('fill', '#F1EEE7');
  square.setAttribute('stroke-width', 1);
  square.setAttribute('stroke', '#CCB');
  svg.appendChild(square);

  if (Robotmaze.SKIN.background) {
    var tile = document.createElementNS(Blockly.SVG_NS, 'image');
    tile.setAttributeNS('http://www.w3.org/1999/xlink', 'xlink:href',
        Robotmaze.SKIN.background);
    tile.setAttribute('height', Robotmaze.MAZE_HEIGHT);
    tile.setAttribute('width', Robotmaze.MAZE_WIDTH);
    tile.setAttribute('x', 0);
    tile.setAttribute('y', 0);
    svg.appendChild(tile);
  }

  if (Robotmaze.SKIN.graph) {
    // Draw the grid lines.
    // The grid lines are offset so that the lines pass through the centre of
    // each square.  A half-pixel offset is also added to as standard SVG
    // practice to avoid blurriness.
    var offset = Robotmaze.SQUARE_SIZE / 2 + 0.5;
    for (var k = 0; k < Robotmaze.ROWS; k++) {
      var h_line = document.createElementNS(Blockly.SVG_NS, 'line');
      h_line.setAttribute('y1', k * Robotmaze.SQUARE_SIZE + offset);
```

```
        h_line.setAttribute('x2', Robotmaze.MAZE_WIDTH);
        h_line.setAttribute('y2', k * Robotmaze.SQUARE_SIZE + offset);
        h_line.setAttribute('stroke', Robotmaze.SKIN.graph);
        h_line.setAttribute('stroke-width', 1);
        svg.appendChild(h_line);
      }
      for (var k = 0; k < Robotmaze.COLS; k++) {
        var v_line = document.createElementNS(Blockly.SVG_NS, 'line');
        v_line.setAttribute('x1', k * Robotmaze.SQUARE_SIZE + offset);
        v_line.setAttribute('x2', k * Robotmaze.SQUARE_SIZE + offset);
        v_line.setAttribute('y2', Robotmaze.MAZE_HEIGHT);
        v_line.setAttribute('stroke', Robotmaze.SKIN.graph);
        v_line.setAttribute('stroke-width', 1);
        svg.appendChild(v_line);
      }
    }

    // Draw the tiles making up the robotmaze map.

    // Return a value of '0' if the specified square is wall or out of bounds,
    // '1' otherwise (empty, start, finish).
    var normalize = function(x, y) {
      if (x < 0 || x >= Robotmaze.COLS || y < 0 || y >= Robotmaze.ROWS) {
        return '0';
      }
      return (Robotmaze.map[y][x] == Robotmaze.SquareType.WALL) ? '0' : '1';
    };

    // Compute and draw the tile for each square.
    var tileId = 0;
    for (var y = 0; y < Robotmaze.ROWS; y++) {
      for (var x = 0; x < Robotmaze.COLS; x++) {
        // Compute the tile shape.
        var tileShape = normalize(x, y) +
            normalize(x, y - 1) +  // North.
            normalize(x + 1, y) +  // West.
            normalize(x, y + 1) +  // South.
            normalize(x - 1, y);   // East.

        // Draw the tile.
        if (!Robotmaze.tile_SHAPES[tileShape]) {
          // Empty square.  Use null0 for large areas, with null1-4 for borders.
          // Add some randomness to avoid large empty spaces.
          if (tileShape == '00000' && Math.random() > 0.3) {
            tileShape = 'null0';
          } else {
            tileShape = 'null' + Math.floor(1 + Math.random() * 4);
          }
        }
        var left = Robotmaze.tile_SHAPES[tileShape][0];
        var top = Robotmaze.tile_SHAPES[tileShape][1];
        // Tile's clipPath element.
        var tileClip = document.createElementNS(Blockly.SVG_NS, 'clipPath');
        tileClip.setAttribute('id', 'tileClipPath' + tileId);
        var clipRect = document.createElementNS(Blockly.SVG_NS, 'rect');
        clipRect.setAttribute('width', Robotmaze.SQUARE_SIZE);
        clipRect.setAttribute('height', Robotmaze.SQUARE_SIZE);

        clipRect.setAttribute('x', x * Robotmaze.SQUARE_SIZE);
        clipRect.setAttribute('y', y * Robotmaze.SQUARE_SIZE);

        tileClip.appendChild(clipRect);
        svg.appendChild(tileClip);
        // Tile sprite.
        var tile = document.createElementNS(Blockly.SVG_NS, 'image');
        tile.setAttributeNS('http://www.w3.org/1999/xlink', 'xlink:href',
            Robotmaze.SKIN.tiles);
        // Position the tile sprite relative to the clipRect.
        tile.setAttribute('height', Robotmaze.SQUARE_SIZE * 4);
        tile.setAttribute('width', Robotmaze.SQUARE_SIZE * 5);
        tile.setAttribute('clip-path', 'url(#tileClipPath' + tileId + ')');
        tile.setAttribute('x', (x - left) * Robotmaze.SQUARE_SIZE);
        tile.setAttribute('y', (y - top) * Robotmaze.SQUARE_SIZE);
        svg.appendChild(tile);
        tileId++;
      }
    }

    // Add finish marker.
    var finishMarker = document.createElementNS(Blockly.SVG_NS, 'image');
    finishMarker.setAttribute('id', 'finish');
    finishMarker.setAttributeNS('http://www.w3.org/1999/xlink', 'xlink:href',
        Robotmaze.SKIN.marker);
    finishMarker.setAttribute('height', 34);
    finishMarker.setAttribute('width', 20);
    svg.appendChild(finishMarker);

    // Pegman's clipPath element, whose (x, y) is reset by Robotmaze.displayPegman
    var pegmanClip = document.createElementNS(Blockly.SVG_NS, 'clipPath');
    pegmanClip.setAttribute('id', 'pegmanClipPath');
    var clipRect = document.createElementNS(Blockly.SVG_NS, 'rect');
    clipRect.setAttribute('id', 'clipRect');
    clipRect.setAttribute('width', Robotmaze.PEGMAN_WIDTH);
    clipRect.setAttribute('height', Robotmaze.PEGMAN_HEIGHT);
    pegmanClip.appendChild(clipRect);
    svg.appendChild(pegmanClip);

    // Add Pegman.
    var pegmanIcon = document.createElementNS(Blockly.SVG_NS, 'image');
    pegmanIcon.setAttribute('id', 'pegman');
    pegmanIcon.setAttributeNS('http://www.w3.org/1999/xlink', 'xlink:href',
        Robotmaze.SKIN.sprite);
    pegmanIcon.setAttribute('height', Robotmaze.PEGMAN_HEIGHT);
    pegmanIcon.setAttribute('width', Robotmaze.PEGMAN_WIDTH * 21); // 49 * 21 = 1029
    pegmanIcon.setAttribute('clip-path', 'url(#pegmanClipPath)');
    svg.appendChild(pegmanIcon);
  };

  /**
   * Initialize Blockly and the robotmaze.  Called on page load.
   */
  Robotmaze.init = function() {
    // Render the Soy template.
    document.body.innerHTML = Robotmaze.soy.start({}, null,
        {lang: BlocklyGames.LANG,
         level: BlocklyGames.LEVEL,
         maxLevel: BlocklyGames.MAX_LEVEL,
         skin: Robotmaze.SKIN_ID,
         html: BlocklyGames.IS_HTML});

    BlocklyInterface.init();

    // Setup the Pegman menu.
    var pegmanImg = document.querySelector('#pegmanButton>img');
    pegmanImg.style.backgroundImage = 'url(' + Robotmaze.SKIN.sprite + ')';
    var pegmanMenu = document.getElementById('pegmanMenu');
    var handlerFactory = function(n) {
      return function() {
        Robotmaze.changePegman(n);
      };
    };
    for (var i = 0; i < Robotmaze.SKINS.length; i++) {
      if (i == Robotmaze.SKIN_ID) {
        continue;
      }
      var div = document.createElement('div');
      var img = document.createElement('img');
      img.src = 'common/1x1.gif';
```

```
      img.style.backgroundImage = 'url(' + Robotmaze.SKINS[i].sprite + ')';
      div.appendChild(img);
      pegmanMenu.appendChild(div);
      Blockly.bindEvent_(div, 'mousedown', null, handlerFactory(i));
    }
    Blockly.bindEvent_(window, 'resize', null, Robotmaze.hidePegmanMenu);
    var pegmanButton = document.getElementById('pegmanButton');
    Blockly.bindEvent_(pegmanButton, 'mousedown', null, Robotmaze.showPegmanMenu);
    var pegmanButtonArrow = document.getElementById('pegmanButtonArrow');
    var arrow = document.createTextNode(Blockly.FieldDropdown.ARROW_CHAR);
    pegmanButtonArrow.appendChild(arrow);

    var rtl = BlocklyGames.isRtl();
    var blocklyDiv = document.getElementById('blockly');
    var visualization = document.getElementById('visualization');
    var onresize = function(e) {
      var top = visualization.offsetTop;
      blocklyDiv.style.top = Math.max(10, top - window.pageYOffset) + 'px';
      blocklyDiv.style.left = rtl ? '10px' : '420px';
      blocklyDiv.style.width = (window.innerWidth - 440) + 'px';
    };
    window.addEventListener('scroll', function() {
      onresize(null);
      Blockly.svgResize(BlocklyGames.workspace);
    });
    window.addEventListener('resize', onresize);
    onresize(null);

    var toolbox = document.getElementById('toolbox');
    // Scale the workspace so level 1 = 1.3, and level 10 = 1.0.
    var scale = 1 + (1 - (BlocklyGames.LEVEL / BlocklyGames.MAX_LEVEL)) / 3;
    BlocklyGames.workspace = Blockly.inject('blockly',
        {'media': 'third-party/blockly/media/',
         'maxBlocks': Robotmaze.MAX_BLOCKS,
         'rtl': rtl,
         'toolbox': toolbox,
         'trashcan': true,
         'zoom': {'startScale': scale}});
    BlocklyGames.workspace.getAudioManager().load(Robotmaze.SKIN.winSound, 'win');
    BlocklyGames.workspace.getAudioManager().load(Robotmaze.SKIN.crashSound, 'fail');
    // Not really needed, there are no user-defined functions or variables.
    Blockly.JavaScript.addReservedWords('moveForward,moveBackward,' +
        'turnRight,turnLeft,isPathForward,isPathRight,isPathBackward,isPathLeft');

    Robotmaze.drawMap();

    var defaultXml =
        '<xml>' +
        '  <block movable="' + (BlocklyGames.LEVEL != 1) + '" ' +
        'type="robotmaze_moveForward" x="70" y="70"></block>' +
        '</xml>';
    BlocklyInterface.loadBlocks(defaultXml, false);

    // Locate the start and finish squares.
    for (var y = 0; y < Robotmaze.ROWS; y++) {
      for (var x = 0; x < Robotmaze.COLS; x++) {
        if (Robotmaze.map[y][x] == Robotmaze.SquareType.START) {
          Robotmaze.start_ = {x: x, y: y};
        } else if (Robotmaze.map[y][x] == Robotmaze.SquareType.FINISH) {
          Robotmaze.finish_ = {x: x, y: y};
        }
      }
    }

    Robotmaze.reset(true);
    BlocklyGames.workspace.addChangeListener(function() {Robotmaze.updateCapacity();});

    document.body.addEventListener('mousemove', Robotmaze.updatePegSpin_, true);

```

```
  BlocklyGames.bindClick('runButton', Robotmaze.runButtonClick);
  BlocklyGames.bindClick('resetButton', Robotmaze.resetButtonClick);

  if (BlocklyGames.LEVEL == 1) {
    // Make connecting blocks easier for beginners.
    Blockly.SNAP_RADIUS *= 2;
    Blockly.CONNECTING_SNAP_RADIUS = Blockly.SNAP_RADIUS;
  }
  if (BlocklyGames.LEVEL == 10) {
    if (!BlocklyGames.loadFromLocalStorage(BlocklyGames.NAME,
                                           BlocklyGames.LEVEL)) {
      // Level 10 gets an introductory modal dialog.
      // Skip the dialog if the user has already won.
      var content = document.getElementById('dialogHelpWallFollow');
      var style = {
        'width': '30%',
        'left': '35%',
        'top': '12em'
      };
      BlocklyDialogs.showDialog(content, null, false, true, style,
          BlocklyDialogs.stopDialogKeyDown);
      BlocklyDialogs.startDialogKeyDown();
      setTimeout(BlocklyDialogs.abortOffer, 5 * 60 * 1000);
    }
  } else {
    // All other levels get interactive help.  But wait 5 seconds for the
    // user to think a bit before they are told what to do.
    setTimeout(function() {
      BlocklyGames.workspace.addChangeListener(Robotmaze.levelHelp);
      Robotmaze.levelHelp();
    }, 5000);
  }

  // Add the spinning Pegman icon to the done dialog.
  // <img id="pegSpin" src="common/1x1.gif">
  var buttonDiv = document.getElementById('dialogDoneButtons');
  var pegSpin = document.createElement('img');
  pegSpin.id = 'pegSpin';
  pegSpin.src = 'common/1x1.gif';
  pegSpin.style.backgroundImage = 'url(' + Robotmaze.SKIN.sprite + ')';
  buttonDiv.parentNode.insertBefore(pegSpin, buttonDiv);

  // Lazy-load the JavaScript interpreter.
  setTimeout(BlocklyInterface.importInterpreter, 1);
  // Lazy-load the syntax-highlighting.
  setTimeout(BlocklyInterface.importPrettify, 1);
};

/**
 * When the workspace changes, update the help as needed.
 * @param {Blockly.Events.Abstract=} opt_event Custom data for event.
 */
Robotmaze.levelHelp = function(opt_event) {
  if (opt_event && opt_event.type == Blockly.Events.UI) {
    // Just a change to highlighting or somesuch.
    return;
  } else if (BlocklyGames.workspace.isDragging()) {
    // Don't change helps during drags.
    return;
  } else if (Robotmaze.result == Robotmaze.ResultType.SUCCESS ||
             BlocklyGames.loadFromLocalStorage(BlocklyGames.NAME,
                                               BlocklyGames.LEVEL)) {
    // The user has already won.  They are just playing around.
    return;
  }
  var rtl = BlocklyGames.isRtl();
  var userBlocks = Blockly.Xml.domToText(
      Blockly.Xml.workspaceToDom(BlocklyGames.workspace));
  var toolbar = BlocklyGames.workspace.flyout_.workspace_.getTopBlocks(true);
  var content = null;
  var origin = null;
  var style = null;
  if (BlocklyGames.LEVEL == 1) {
    if (BlocklyGames.workspace.getAllBlocks().length < 2) {
      content = document.getElementById('dialogHelpStack');
      style = {'width': '370px', 'top': '130px'};
      style[rtl ? 'right' : 'left'] = '215px';
      origin = toolbar[0].getSvgRoot();
    } else {
      var topBlocks = BlocklyGames.workspace.getTopBlocks(true);
      if (topBlocks.length > 1) {
        var xml = [
            '<xml>',
              '<block type="robotmaze_moveForward" x="10" y="10">',
                '<next>',
                  '<block type="robotmaze_moveForward"></block>',
                '</next>',
              '</block>',
            '</xml>'];
        BlocklyInterface.injectReadonly('sampleOneTopBlock', xml);
        content = document.getElementById('dialogHelpOneTopBlock');
        style = {'width': '360px', 'top': '120px'};
        style[rtl ? 'right' : 'left'] = '225px';
        origin = topBlocks[0].getSvgRoot();
      } else if (Robotmaze.result == Robotmaze.ResultType.UNSET) {
        // Show run help dialog.
        content = document.getElementById('dialogHelpRun');
        style = {'width': '360px', 'top': '410px'};
        style[rtl ? 'right' : 'left'] = '400px';
        origin = document.getElementById('runButton');
      }
    }
  } else if (BlocklyGames.LEVEL == 2) {
    if (Robotmaze.result != Robotmaze.ResultType.UNSET &&
        document.getElementById('runButton').style.display == 'none') {
      content = document.getElementById('dialogHelpReset');
      style = {'width': '360px', 'top': '410px'};
      style[rtl ? 'right' : 'left'] = '400px';
      origin = document.getElementById('resetButton');
    }
  } else if (BlocklyGames.LEVEL == 3) {
    if (userBlocks.indexOf('robotmaze_forever') == -1) {
      if (BlocklyGames.workspace.remainingCapacity() == 0) {
        content = document.getElementById('dialogHelpCapacity');
        style = {'width': '430px', 'top': '310px'};
        style[rtl ? 'right' : 'left'] = '50px';
        origin = document.getElementById('capacityBubble');
      } else {
        content = document.getElementById('dialogHelpRepeat');
        style = {'width': '360px', 'top': '360px'};
        style[rtl ? 'right' : 'left'] = '425px';
        origin = toolbar[3].getSvgRoot();
      }
    }
  } else if (BlocklyGames.LEVEL == 4) {
    if (BlocklyGames.workspace.remainingCapacity() == 0 &&
        (userBlocks.indexOf('robotmaze_forever') == -1 ||
         BlocklyGames.workspace.getTopBlocks(false).length > 1)) {
      content = document.getElementById('dialogHelpCapacity');
      style = {'width': '430px', 'top': '310px'};
      style[rtl ? 'right' : 'left'] = '50px';
      origin = document.getElementById('capacityBubble');
    } else {
      var showHelp = true;
      // Only show help if there is not a loop with two nested blocks.
      var blocks = BlocklyGames.workspace.getAllBlocks();
      for (var i = 0; i < blocks.length; i++) {
        var block = blocks[i];
```

```
          if (block.type != 'robotmaze_forever') {
            continue;
          }
          var j = 0;
          while (block) {
            var kids = block.getChildren();
            block = kids.length ? kids[0] : null;
            j++;
          }
          if (j > 2) {
            showHelp = false;
            break;
          }
        }
        if (showHelp) {
          content = document.getElementById('dialogHelpRepeatMany');
          style = {'width': '360px', 'top': '360px'};
          style[rtl ? 'right' : 'left'] = '425px';
          origin = toolbar[3].getSvgRoot();
        }
      }
    } else if (BlocklyGames.LEVEL == 5) {
      if (Robotmaze.SKIN_ID == 0 && !Robotmaze.showPegmanMenu.activatedOnce) {
        content = document.getElementById('dialogHelpSkins');
        style = {'width': '360px', 'top': '60px'};
        style[rtl ? 'left' : 'right'] = '20px';
        origin = document.getElementById('pegmanButton');
      }
    } else if (BlocklyGames.LEVEL == 6) {
      if (userBlocks.indexOf('robotmaze_if') == -1) {
        content = document.getElementById('dialogHelpIf');
        style = {'width': '360px', 'top': '430px'};
        style[rtl ? 'right' : 'left'] = '425px';
        origin = toolbar[4].getSvgRoot();
      }
    } else if (BlocklyGames.LEVEL == 7) {
      if (!Robotmaze.levelHelp.initialized7_) {
        // Create fake dropdown.
        var span = document.createElement('span');
        span.className = 'helpMenuFake';
        var options =
            [BlocklyGames.getMsg('Robotmaze_pathAhead'),
             BlocklyGames.getMsg('Robotmaze_pathLeft'),
             BlocklyGames.getMsg('Robotmaze_pathRight')];
        var prefix = Blockly.utils.commonWordPrefix(options);
        var suffix = Blockly.utils.commonWordSuffix(options);
        if (suffix) {
          var option = options[0].slice(prefix, -suffix);
        } else {
          var option = options[0].substring(prefix);
        }
        // Add dropdown arrow: "option ▾" (LTR) or "▾ אופציה" (RTL)
        span.textContent = option + ' ' + Blockly.FieldDropdown.ARROW_CHAR;
        // Inject fake dropdown into message.
        var container = document.getElementById('helpMenuText');
        var msg = container.textContent;
        container.textContent = '';
        var parts = msg.split(/%\d/);
        for (var i = 0; i < parts.length; i++) {
          container.appendChild(document.createTextNode(parts[i]));
          if (i != parts.length - 1) {
            container.appendChild(span.cloneNode(true));
          }
        }
        Robotmaze.levelHelp.initialized7_ = true;
      }
      // The hint says to change from 'ahead', but keep the hint visible
      // until the user chooses 'right'.
      if (userBlocks.indexOf('isPathRight') == -1) {
```

```
      content = document.getElementById('dialogHelpMenu');
      style = {'width': '360px', 'top': '430px'};
      style[rtl ? 'right' : 'left'] = '425px';
      origin = toolbar[4].getSvgRoot();
    }
  } else if (BlocklyGames.LEVEL == 9) {
    if (userBlocks.indexOf('robotmaze_ifElse') == -1) {
      content = document.getElementById('dialogHelpIfElse');
      style = {'width': '360px', 'top': '305px'};
      style[rtl ? 'right' : 'left'] = '425px';
      origin = toolbar[5].getSvgRoot();
    }
  }
  if (content) {
    if (content.parentNode != document.getElementById('dialog')) {
      BlocklyDialogs.showDialog(content, origin, true, false, style, null);
    }
  } else {
    BlocklyDialogs.hideDialog(false);
  }
};

/**
 * Reload with a different Pegman skin.
 * @param {number} newSkin ID of new skin.
 */
Robotmaze.changePegman = function(newSkin) {
  Robotmaze.saveToStorage();
  window.location = window.location.protocol + '//' +
      window.location.host + window.location.pathname +
      '?lang=' + BlocklyGames.LANG + '&level=' + BlocklyGames.LEVEL +
      '&skin=' + newSkin;
};

/**
 * Save the blocks for a one-time reload.
 */
Robotmaze.saveToStorage = function() {
  // MSIE 11 does not support sessionStorage on file:// URLs.
  if (typeof Blockly != undefined && window.sessionStorage) {
    var xml = Blockly.Xml.workspaceToDom(BlocklyGames.workspace);
    var text = Blockly.Xml.domToText(xml);
    window.sessionStorage.loadOnceBlocks = text;
  }
};

/**
 * Display the Pegman skin-change menu.
 * @param {!Event} e Mouse, touch, or resize event.
 */
Robotmaze.showPegmanMenu = function(e) {
  var menu = document.getElementById('pegmanMenu');
  if (menu.style.display == 'block') {
    // Menu is already open.  Close it.
    Robotmaze.hidePegmanMenu(e);
    return;
  }
  // Prevent double-clicks or double-taps.
  if (BlocklyInterface.eventSpam(e)) {
    return;
  }
  var button = document.getElementById('pegmanButton');
  button.classList.add('buttonHover');
  menu.style.top = (button.offsetTop + button.offsetHeight) + 'px';
  menu.style.left = button.offsetLeft + 'px';
  menu.style.display = 'block';
  Robotmaze.pegmanMenuMouse_ =
      Blockly.bindEvent_(document.body, 'mousedown', null, Robotmaze.hidePegmanMenu);
  // Close the skin-changing hint if open.
  var hint = document.getElementById('dialogHelpSkins');
  if (hint && hint.className != 'dialogHiddenContent') {
    BlocklyDialogs.hideDialog(false);
  }
  Robotmaze.showPegmanMenu.activatedOnce = true;
};

/**
 * Hide the Pegman skin-change menu.
 * @param {!Event} e Mouse, touch, or resize event.
 */
Robotmaze.hidePegmanMenu = function(e) {
  // Prevent double-clicks or double-taps.
  if (BlocklyInterface.eventSpam(e)) {
    return;
  }
  document.getElementById('pegmanMenu').style.display = 'none';
  document.getElementById('pegmanButton').classList.remove('buttonHover');
  if (Robotmaze.pegmanMenuMouse_) {
    Blockly.unbindEvent_(Robotmaze.pegmanMenuMouse_);
    delete Robotmaze.pegmanMenuMouse_;
  }
};

/**
 * Reset the robotmaze to the start position and kill any pending animation tasks.
 * @param {boolean} first True if an opening animation is to be played.
 */
Robotmaze.reset = function(first) {
  // Kill all tasks.
  for (var i = 0; i < Robotmaze.pidList.length; i++) {
    window.clearTimeout(Robotmaze.pidList[i]);
  }
  Robotmaze.pidList = [];

  // Move Pegman into position.
  Robotmaze.pegmanX = Robotmaze.start_.x;
  Robotmaze.pegmanY = Robotmaze.start_.y;

  if (first) {
    Robotmaze.pegmanD = Robotmaze.startDirection + 1;
    Robotmaze.scheduleFinish(false);
    Robotmaze.pidList.push(setTimeout(function() {
      Robotmaze.stepSpeed = 100;
      Robotmaze.schedule([Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4],
                    [Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4 - 4]);
      Robotmaze.pegmanD++;
    }, Robotmaze.stepSpeed * 5));
  } else {
    Robotmaze.pegmanD = Robotmaze.startDirection;
    Robotmaze.displayPegman(Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4);
  }

  // Move the finish icon into position.
  var finishIcon = document.getElementById('finish');
  finishIcon.setAttribute('x', Robotmaze.SQUARE_SIZE * (Robotmaze.finish_.x + 0.5) -
      finishIcon.getAttribute('width') / 2);
  finishIcon.setAttribute('y', Robotmaze.SQUARE_SIZE * (Robotmaze.finish_.y + 0.6) -
      finishIcon.getAttribute('height'));

  // Make 'look' icon invisible and promote to top.
  var lookIcon = document.getElementById('look');
  lookIcon.style.display = 'none';
  lookIcon.parentNode.appendChild(lookIcon);
  var paths = lookIcon.getElementsByTagName('path');
  for (var i = 0, path; (path = paths[i]); i++) {
    path.setAttribute('stroke', Robotmaze.SKIN.look);
  }
};
```

```

/**
 * Click the run button.  Start the program.
 * @param {!Event} e Mouse or touch event.
 */
Robotmaze.runButtonClick = function(e) {
  // Prevent double-clicks or double-taps.
  if (BlocklyInterface.eventSpam(e)) {
    return;
  }
  BlocklyDialogs.hideDialog(false);
  // Only allow a single top block on level 1.
  if (BlocklyGames.LEVEL == 1 &&
      BlocklyGames.workspace.getTopBlocks(false).length > 1 &&
      Robotmaze.result != Robotmaze.ResultType.SUCCESS &&
      !BlocklyGames.loadFromLocalStorage(BlocklyGames.NAME,
                                         BlocklyGames.LEVEL)) {
    Robotmaze.levelHelp();
    return;
  }
  var runButton = document.getElementById('runButton');
  var resetButton = document.getElementById('resetButton');
  // Ensure that Reset button is at least as wide as Run button.
  if (!resetButton.style.minWidth) {
    resetButton.style.minWidth = runButton.offsetWidth + 'px';
  }
  runButton.style.display = 'none';
  resetButton.style.display = 'inline';
  Robotmaze.reset(false);
  Robotmaze.execute();
};

/**
 * Updates the document's 'capacity' element with a message
 * indicating how many more blocks are permitted.  The capacity
 * is retrieved from BlocklyGames.workspace.remainingCapacity().
 */
Robotmaze.updateCapacity = function() {
  var cap = BlocklyGames.workspace.remainingCapacity();
  var p = document.getElementById('capacity');
  if (cap == Infinity) {
    p.style.display = 'none';
  } else {
    p.style.display = 'inline';
    p.innerHTML = '';
    cap = Number(cap);
    var capSpan = document.createElement('span');
    capSpan.className = 'capacityNumber';
    capSpan.appendChild(document.createTextNode(cap));
    if (cap == 0) {
      var msg = BlocklyGames.getMsg('Robotmaze_capacity0');
    } else if (cap == 1) {
      var msg = BlocklyGames.getMsg('Robotmaze_capacity1');
    } else {
      var msg = BlocklyGames.getMsg('Robotmaze_capacity2');
    }
    var parts = msg.split(/%\d/);
    for (var i = 0; i < parts.length; i++) {
      p.appendChild(document.createTextNode(parts[i]));
      if (i != parts.length - 1) {
        p.appendChild(capSpan.cloneNode(true));
      }
    }
  }
};

/**
 * Click the reset button.  Reset the robotmaze.
 * @param {!Event} e Mouse or touch event.
 */
Robotmaze.resetButtonClick = function(e) {
  // Prevent double-clicks or double-taps.
  if (BlocklyInterface.eventSpam(e)) {
    return;
  }
  var runButton = document.getElementById('runButton');
  runButton.style.display = 'inline';
  document.getElementById('resetButton').style.display = 'none';
  BlocklyGames.workspace.highlightBlock(null);
  Robotmaze.reset(false);
  Robotmaze.levelHelp();
};

/**
 * Inject the Robotmaze API into a JavaScript interpreter.
 * @param {!Interpreter} interpreter The JS Interpreter.
 * @param {!Interpreter.Object} scope Global scope.
 */
Robotmaze.initInterpreter = function(interpreter, scope) {
  // API
  var wrapper;
  wrapper = function(id) {
    Robotmaze.move(0, id);
  };
  interpreter.setProperty(scope, 'moveForward',
      interpreter.createNativeFunction(wrapper));
  wrapper = function(id) {
    Robotmaze.move(2, id);
  };
  interpreter.setProperty(scope, 'moveBackward',
      interpreter.createNativeFunction(wrapper));
  wrapper = function(id) {
    Robotmaze.turn(0, id);
  };
  interpreter.setProperty(scope, 'turnLeft',
      interpreter.createNativeFunction(wrapper));
  wrapper = function(id) {
    Robotmaze.turn(1, id);
  };
  interpreter.setProperty(scope, 'turnRight',
      interpreter.createNativeFunction(wrapper));
  wrapper = function(id) {
    return Robotmaze.isPath(0, id);
  };
  interpreter.setProperty(scope, 'isPathForward',
      interpreter.createNativeFunction(wrapper));
  wrapper = function(id) {
    return Robotmaze.isPath(1, id);
  };
  interpreter.setProperty(scope, 'isPathRight',
      interpreter.createNativeFunction(wrapper));
  wrapper = function(id) {
    return Robotmaze.isPath(2, id);
  };
  interpreter.setProperty(scope, 'isPathBackward',
      interpreter.createNativeFunction(wrapper));
  wrapper = function(id) {
    return Robotmaze.isPath(3, id);
  };
  interpreter.setProperty(scope, 'isPathLeft',
      interpreter.createNativeFunction(wrapper));
  wrapper = function() {
    return Robotmaze.notDone();
  };
  interpreter.setProperty(scope, 'notDone',
      interpreter.createNativeFunction(wrapper));
};

```

```
/**
 * Execute the user's code.  Heaven help us...
 */
Robotmaze.execute = function() {
  if (!('Interpreter' in window)) {
    // Interpreter lazy loads and hasn't arrived yet.  Try again later.
    setTimeout(Robotmaze.execute, 250);
    return;
  }

  Robotmaze.log = [];
  Blockly.selected && Blockly.selected.unselect();
  var code = Blockly.JavaScript.workspaceToCode(BlocklyGames.workspace);
  Robotmaze.result = Robotmaze.ResultType.UNSET;
  var interpreter = new Interpreter(code, Robotmaze.initInterpreter);

  // Try running the user's code.  There are four possible outcomes:
  // 1. If pegman reaches the finish [SUCCESS], true is thrown.
  // 2. If the program is terminated due to running too long [TIMEOUT],
  //    false is thrown.
  // 3. If another error occurs [ERROR], that error is thrown.
  // 4. If the program ended normally but without solving the robotmaze [FAILURE],
  //    no error or exception is thrown.
  try {
    var ticks = 10000;  // 10k ticks runs Pegman for about 8 minutes.
    while (interpreter.step()) {
      if (ticks-- == 0) {
        throw Infinity;
      }
    }
    Robotmaze.result = Robotmaze.notDone() ?
        Robotmaze.ResultType.FAILURE : Robotmaze.ResultType.SUCCESS;
  } catch (e) {
    // A boolean is thrown for normal termination.
    // Abnormal termination is a user error.
    if (e === Infinity) {
      Robotmaze.result = Robotmaze.ResultType.TIMEOUT;
    } else if (e === false) {
      Robotmaze.result = Robotmaze.ResultType.ERROR;
    } else {
      // Syntax error, can't happen.
      Robotmaze.result = Robotmaze.ResultType.ERROR;
      alert(e);
    }
  }

  // Fast animation if execution is successful.  Slow otherwise.
  if (Robotmaze.result == Robotmaze.ResultType.SUCCESS) {
    Robotmaze.stepSpeed = 100;
    Robotmaze.log.push(['finish', null]);
  } else {
    Robotmaze.stepSpeed = 150;
  }

  // Robotmaze.log now contains a transcript of all the user's actions.
  // Reset the robotmaze and animate the transcript.
  Robotmaze.reset(false);
  Robotmaze.pidList.push(setTimeout(Robotmaze.animate, 100));
};

/**
 * Iterate through the recorded path and animate pegman's actions.
 */
Robotmaze.animate = function() {
  var action = Robotmaze.log.shift();
  if (!action) {
    BlocklyInterface.highlight(null);
    Robotmaze.levelHelp();
    return;
  }
  BlocklyInterface.highlight(action[1]);

  switch (action[0]) {
    case 'north':
      Robotmaze.schedule([Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4],
                         [Robotmaze.pegmanX, Robotmaze.pegmanY - 1, Robotmaze.pegmanD * 4]);
      Robotmaze.pegmanY--;
      break;
    case 'east':
      Robotmaze.schedule([Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4],
                         [Robotmaze.pegmanX + 1, Robotmaze.pegmanY, Robotmaze.pegmanD * 4]);
      Robotmaze.pegmanX++;
      break;
    case 'south':
      Robotmaze.schedule([Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4],
                         [Robotmaze.pegmanX, Robotmaze.pegmanY + 1, Robotmaze.pegmanD * 4]);
      Robotmaze.pegmanY++;
      break;
    case 'west':
      Robotmaze.schedule([Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4],
                         [Robotmaze.pegmanX - 1, Robotmaze.pegmanY, Robotmaze.pegmanD * 4]);
      Robotmaze.pegmanX--;
      break;
    case 'look_north':
      Robotmaze.scheduleLook(Robotmaze.DirectionType.NORTH);
      break;
    case 'look_east':
      Robotmaze.scheduleLook(Robotmaze.DirectionType.EAST);
      break;
    case 'look_south':
      Robotmaze.scheduleLook(Robotmaze.DirectionType.SOUTH);
      break;
    case 'look_west':
      Robotmaze.scheduleLook(Robotmaze.DirectionType.WEST);
      break;
    case 'fail_forward':
      Robotmaze.scheduleFail(true);
      break;
    case 'fail_backward':
      Robotmaze.scheduleFail(false);
      break;
    case 'left':
      Robotmaze.schedule([Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4],
                         [Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4 - 4]);
      Robotmaze.pegmanD = Robotmaze.constrainDirection4(Robotmaze.pegmanD - 1);
      break;
    case 'right':
      Robotmaze.schedule([Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4],
                         [Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4 + 4]);
      Robotmaze.pegmanD = Robotmaze.constrainDirection4(Robotmaze.pegmanD + 1);
      break;
    case 'finish':
      Robotmaze.scheduleFinish(true);
      BlocklyInterface.saveToLocalStorage();
      setTimeout(BlocklyDialogs.congratulations, 1000);
  }

  Robotmaze.pidList.push(setTimeout(Robotmaze.animate, Robotmaze.stepSpeed * 5));
};

/**
 * Point the congratulations Pegman to face the mouse.
 * @param {Event} e Mouse move event.
 * @private
 */
Robotmaze.updatePegSpin_ = function(e) {
  if (document.getElementById('dialogDone').className ==
      'dialogHiddenContent') {
```

```
      return;
    }
    var pegSpin = document.getElementById('pegSpin');
    var bBox = BlocklyDialogs.getBBox_(pegSpin);
    var x = bBox.x + bBox.width / 2 - window.pageXOffset;
    var y = bBox.y + bBox.height / 2 - window.pageYOffset;
    var dx = e.clientX - x;
    var dy = e.clientY - y;
    var angle = Math.atan(dy / dx);
    // Convert from radians to degrees because I suck at math.
    angle = angle / Math.PI * 180;
    // 0: North, 90: East, 180: South, 270: West.
    if (dx > 0) {
      angle += 90;
    } else {
      angle += 270;
    }
    // Divide into 16 quads.
    var quad = Math.round(angle / 360 * 16);
    if (quad == 16) {
      quad = 15;
    }
    // Display correct Pegman sprite.
    pegSpin.style.backgroundPosition = (-quad * Robotmaze.PEGMAN_WIDTH) + 'px 0px';
  };

  /**
   * Schedule the animations for a move or turn.
   * @param {!Array.<number>} startPos X, Y and direction starting points.
   * @param {!Array.<number>} endPos X, Y and direction ending points.
   */
  Robotmaze.schedule = function(startPos, endPos) {
    var deltas = [(endPos[0] - startPos[0]) / 4,
                  (endPos[1] - startPos[1]) / 4,
                  (endPos[2] - startPos[2]) / 4];
    Robotmaze.displayPegman(startPos[0] + deltas[0],
                            startPos[1] + deltas[1],
                            Robotmaze.constrainDirection16(startPos[2] + deltas[2]));
    Robotmaze.pidList.push(setTimeout(function() {
        Robotmaze.displayPegman(startPos[0] + deltas[0] * 2,
            startPos[1] + deltas[1] * 2,
            Robotmaze.constrainDirection16(startPos[2] + deltas[2] * 2));
      }, Robotmaze.stepSpeed));
    Robotmaze.pidList.push(setTimeout(function() {
        Robotmaze.displayPegman(startPos[0] + deltas[0] * 3,
            startPos[1] + deltas[1] * 3,
            Robotmaze.constrainDirection16(startPos[2] + deltas[2] * 3));
      }, Robotmaze.stepSpeed * 2));
    Robotmaze.pidList.push(setTimeout(function() {
        Robotmaze.displayPegman(endPos[0], endPos[1],
            Robotmaze.constrainDirection16(endPos[2]));
      }, Robotmaze.stepSpeed * 3));
  };

  /**
   * Schedule the animations and sounds for a failed move.
   * @param {boolean} forward True if forward, false if backward.
   */
  Robotmaze.scheduleFail = function(forward) {
    var deltaX = 0;
    var deltaY = 0;
    switch (Robotmaze.pegmanD) {
      case Robotmaze.DirectionType.NORTH:
        deltaY = -1;
        break;
      case Robotmaze.DirectionType.EAST:
        deltaX = 1;
        break;
      case Robotmaze.DirectionType.SOUTH:
        deltaY = 1;
        break;
      case Robotmaze.DirectionType.WEST:
        deltaX = -1;
        break;
    }
    if (!forward) {
      deltaX = -deltaX;
      deltaY = -deltaY;
    }
    if (Robotmaze.SKIN.crashType == Robotmaze.CRASH_STOP) {
      // Bounce bounce.
      deltaX /= 4;
      deltaY /= 4;
      var direction16 = Robotmaze.constrainDirection16(Robotmaze.pegmanD * 4);
      Robotmaze.displayPegman(Robotmaze.pegmanX + deltaX,
                              Robotmaze.pegmanY + deltaY,
                              direction16);
      BlocklyGames.workspace.getAudioManager().play('fail', 0.5);
      Robotmaze.pidList.push(setTimeout(function() {
        Robotmaze.displayPegman(Robotmaze.pegmanX,
                                Robotmaze.pegmanY,
                                direction16);
        }, Robotmaze.stepSpeed));
      Robotmaze.pidList.push(setTimeout(function() {
        Robotmaze.displayPegman(Robotmaze.pegmanX + deltaX,
                                Robotmaze.pegmanY + deltaY,
                                direction16);
        BlocklyGames.workspace.getAudioManager().play('fail', 0.5);
      }, Robotmaze.stepSpeed * 2));
      Robotmaze.pidList.push(setTimeout(function() {
          Robotmaze.displayPegman(Robotmaze.pegmanX, Robotmaze.pegmanY, direction16);
        }, Robotmaze.stepSpeed * 3));
    } else {
      // Add a small random delta away from the grid.
      var deltaZ = (Math.random() - 0.5) * 10;
      var deltaD = (Math.random() - 0.5) / 2;
      deltaX += (Math.random() - 0.5) / 4;
      deltaY += (Math.random() - 0.5) / 4;
      deltaX /= 8;
      deltaY /= 8;
      var acceleration = 0;
      if (Robotmaze.SKIN.crashType == Robotmaze.CRASH_FALL) {
        acceleration = 0.01;
      }
      Robotmaze.pidList.push(setTimeout(function() {
        BlocklyGames.workspace.getAudioManager().play('fail', 0.5);
      }, Robotmaze.stepSpeed * 2));
      var setPosition = function(n) {
        return function() {
          var direction16 = Robotmaze.constrainDirection16(Robotmaze.pegmanD * 4 +
                                                            deltaD * n);
          Robotmaze.displayPegman(Robotmaze.pegmanX + deltaX * n,
                                  Robotmaze.pegmanY + deltaY * n,
                                  direction16,
                                  deltaZ * n);
          deltaY += acceleration;
        };
      };
      // 100 frames should get Pegman offscreen.
      for (var i = 1; i < 100; i++) {
        Robotmaze.pidList.push(setTimeout(setPosition(i),
            Robotmaze.stepSpeed * i / 2));
      }
    }
  };

  /**
   * Schedule the animations and sound for a victory dance.
```

```javascript
 * @param {boolean} sound Play the victory sound.
 */
Robotmaze.scheduleFinish = function(sound) {
  var direction16 = Robotmaze.constrainDirection16(Robotmaze.pegmanD * 4);
  Robotmaze.displayPegman(Robotmaze.pegmanX, Robotmaze.pegmanY, 16);
  if (sound) {
    BlocklyGames.workspace.getAudioManager().play('win', 0.5);
  }
  Robotmaze.stepSpeed = 150;  // Slow down victory animation a bit.
  Robotmaze.pidList.push(setTimeout(function() {
    Robotmaze.displayPegman(Robotmaze.pegmanX, Robotmaze.pegmanY, 18);
    }, Robotmaze.stepSpeed));
  Robotmaze.pidList.push(setTimeout(function() {
    Robotmaze.displayPegman(Robotmaze.pegmanX, Robotmaze.pegmanY, 16);
    }, Robotmaze.stepSpeed * 2));
  Robotmaze.pidList.push(setTimeout(function() {
      Robotmaze.displayPegman(Robotmaze.pegmanX, Robotmaze.pegmanY, direction16);
    }, Robotmaze.stepSpeed * 3));
};

/**
 * Display Pegman at the specified location, facing the specified direction.
 * @param {number} x Horizontal grid (or fraction thereof).
 * @param {number} y Vertical grid (or fraction thereof).
 * @param {number} d Direction (0 - 15) or dance (16 - 17).
 * @param {number=} opt_angle Optional angle (in degrees) to rotate Pegman.
 */
Robotmaze.displayPegman = function(x, y, d, opt_angle) {
  var pegmanIcon = document.getElementById('pegman');
  pegmanIcon.setAttribute('x',
      x * Robotmaze.SQUARE_SIZE - d * Robotmaze.PEGMAN_WIDTH + 1);
  pegmanIcon.setAttribute('y',
      Robotmaze.SQUARE_SIZE * (y + 0.5) - Robotmaze.PEGMAN_HEIGHT / 2 - 8);
  if (opt_angle) {
    pegmanIcon.setAttribute('transform', 'rotate(' + opt_angle + ', ' +
        (x * Robotmaze.SQUARE_SIZE + Robotmaze.SQUARE_SIZE / 2) + ', ' +
        (y * Robotmaze.SQUARE_SIZE + Robotmaze.SQUARE_SIZE / 2) + ')');
  } else {
    pegmanIcon.setAttribute('transform', 'rotate(0, 0, 0)');
  }

  var clipRect = document.getElementById('clipRect');
  clipRect.setAttribute('x', x * Robotmaze.SQUARE_SIZE + 1);
  clipRect.setAttribute('y', pegmanIcon.getAttribute('y'));
};

/**
 * Display the look icon at Pegman's current location,
 * in the specified direction.
 * @param {!Robotmaze.DirectionType} d Direction (0 - 3).
 */
Robotmaze.scheduleLook = function(d) {
  var x = Robotmaze.pegmanX;
  var y = Robotmaze.pegmanY;
  switch (d) {
    case Robotmaze.DirectionType.NORTH:
      x += 0.5;
      break;
    case Robotmaze.DirectionType.EAST:
      x += 1;
      y += 0.5;
      break;
    case Robotmaze.DirectionType.SOUTH:
      x += 0.5;
      y += 1;
      break;
    case Robotmaze.DirectionType.WEST:
      y += 0.5;
      break;
  }
  x *= Robotmaze.SQUARE_SIZE;
  y *= Robotmaze.SQUARE_SIZE;
  var deg = d * 90 - 45;

  var lookIcon = document.getElementById('look');
  lookIcon.setAttribute('transform',
      'translate(' + x + ', ' + y + ') ' +
      'rotate(' + deg + ' 0 0) scale(.4)');
  var paths = lookIcon.getElementsByTagName('path');
  lookIcon.style.display = 'inline';
  for (var i = 0, path; (path = paths[i]); i++) {
    Robotmaze.scheduleLookStep(path, Robotmaze.stepSpeed * i);
  }
};

/**
 * Schedule one of the 'look' icon's waves to appear, then disappear.
 * @param {!Element} path Element to make appear.
 * @param {number} delay Milliseconds to wait before making wave appear.
 */
Robotmaze.scheduleLookStep = function(path, delay) {
  Robotmaze.pidList.push(setTimeout(function() {
    path.style.display = 'inline';
    setTimeout(function() {
      path.style.display = 'none';
    }, Robotmaze.stepSpeed * 2);
  }, delay));
};

/**
 * Keep the direction within 0-3, wrapping at both ends.
 * @param {number} d Potentially out-of-bounds direction value.
 * @return {number} Legal direction value.
 */
Robotmaze.constrainDirection4 = function(d) {
  d = Math.round(d) % 4;
  if (d < 0) {
    d += 4;
  }
  return d;
};

/**
 * Keep the direction within 0-15, wrapping at both ends.
 * @param {number} d Potentially out-of-bounds direction value.
 * @return {number} Legal direction value.
 */
Robotmaze.constrainDirection16 = function(d) {
  d = Math.round(d) % 16;
  if (d < 0) {
    d += 16;
  }
  return d;
};

// Core functions.

/**
 * Attempt to move pegman forward or backward.
 * @param {number} direction Direction to move (0 = forward, 2 = backward).
 * @param {string} id ID of block that triggered this action.
 * @throws {true} If the end of the robotmaze is reached.
 * @throws {false} If Pegman collides with a wall.
 */
Robotmaze.move = function(direction, id) {
  if (!Robotmaze.isPath(direction, null)) {
    Robotmaze.log.push(['fail_' + (direction ? 'backward' : 'forward'), id]);
    throw false;
  }
  // If moving backward, flip the effective direction.
  var effectiveDirection = Robotmaze.pegmanD + direction;
  var command;
  switch (Robotmaze.constrainDirection4(effectiveDirection)) {
    case Robotmaze.DirectionType.NORTH:
      Robotmaze.pegmanY--;
      command = 'north';
      break;
    case Robotmaze.DirectionType.EAST:
      Robotmaze.pegmanX++;
      command = 'east';
      break;
    case Robotmaze.DirectionType.SOUTH:
      Robotmaze.pegmanY++;
      command = 'south';
      break;
    case Robotmaze.DirectionType.WEST:
      Robotmaze.pegmanX--;
      command = 'west';
      break;
  }
  Robotmaze.log.push([command, id]);
};

/**
 * Turn pegman left or right.
 * @param {number} direction Direction to turn (0 = left, 1 = right).
 * @param {string} id ID of block that triggered this action.
 */
Robotmaze.turn = function(direction, id) {
  if (direction) {
    // Right turn (clockwise).
    Robotmaze.pegmanD++;
    Robotmaze.log.push(['right', id]);
  } else {
    // Left turn (counterclockwise).
    Robotmaze.pegmanD--;
    Robotmaze.log.push(['left', id]);
  }
  Robotmaze.pegmanD = Robotmaze.constrainDirection4(Robotmaze.pegmanD);
};

/**
 * Is there a path next to pegman?
 * @param {number} direction Direction to look
 *     (0 = forward, 1 = right, 2 = backward, 3 = left).
 * @param {?string} id ID of block that triggered this action.
 *     Null if called as a helper function in Robotmaze.move().
 * @return {boolean} True if there is a path.
 */
Robotmaze.isPath = function(direction, id) {
  var effectiveDirection = Robotmaze.pegmanD + direction;
  var square;
  var command;
  switch (Robotmaze.constrainDirection4(effectiveDirection)) {
    case Robotmaze.DirectionType.NORTH:
      square = Robotmaze.map[Robotmaze.pegmanY - 1] &&
          Robotmaze.map[Robotmaze.pegmanY - 1][Robotmaze.pegmanX];
      command = 'look_north';
      break;
    case Robotmaze.DirectionType.EAST:
      square = Robotmaze.map[Robotmaze.pegmanY][Robotmaze.pegmanX + 1];
      command = 'look_east';
      break;
    case Robotmaze.DirectionType.SOUTH:
      square = Robotmaze.map[Robotmaze.pegmanY + 1] &&
          Robotmaze.map[Robotmaze.pegmanY + 1][Robotmaze.pegmanX];
      command = 'look_south';
      break;
    case Robotmaze.DirectionType.WEST:
      square = Robotmaze.map[Robotmaze.pegmanY][Robotmaze.pegmanX - 1];
      command = 'look_west';
      break;
  }
  if (id) {
    Robotmaze.log.push([command, id]);
  }
  return square !== Robotmaze.SquareType.WALL && square !== undefined;
};

/**
 * Is the player at the finish marker?
 * @return {boolean} True if not done, false if done.
 */
Robotmaze.notDone = function() {
  return Robotmaze.pegmanX != Robotmaze.finish_.x || Robotmaze.pegmanY !=
  Robotmaze.finish_.y;
};

window.addEventListener('load', Robotmaze.init);
```