

```

1  /**
2   * Blockly Games: Robotmaze
3   *
4   * Copyright 2012 Google Inc.
5   * https://github.com/google/blockly-games
6   *
7   * Licensed under the Apache License, Version 2.0 (the "License");
8   * you may not use this file except in compliance with the License.
9   * You may obtain a copy of the License at
10  *
11  * http://www.apache.org/licenses/LICENSE-2.0
12  *
13  * Unless required by applicable law or agreed to in writing, software
14  * distributed under the License is distributed on an "AS IS" BASIS,
15  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
16  * See the License for the specific language governing permissions and
17  * limitations under the License.
18  */
19
20 /**
21  * @fileoverview JavaScript for Blockly's Robotmaze application.
22  * @author fraser@google.com (Neil Fraser)
23  */
24 'use strict';
25
26 goog.provide('Robotmaze');
27
28 goog.require('Blockly.FieldDropdown');
29 goog.require('BlocklyDialogs');
30 goog.require('BlocklyGames');
31 goog.require('BlocklyInterface');
32 goog.require('Maze.Blocks');
33 goog.require('Robotmaze.soy');
34
35 BlocklyGames.NAME = 'robotmaze';
36
37 /**
38  * Go to the next level. Add skin parameter.
39  * @suppress {duplicate}
40  */
41 BlocklyInterface.nextLevel = function() {
42   if (BlocklyGames.LEVEL < BlocklyGames.MAX_LEVEL) {
43     window.location = window.location.protocol + '//' +
44       window.location.host + window.location.pathname +
45       '?lang=' + BlocklyGames.LANG + '&level=' + (BlocklyGames.LEVEL + 1) +
46       '&skin=' + Robotmaze.SKIN_ID;
47   } else {
48     BlocklyInterface.indexPage();
49   }
50 };
51
52 Robotmaze.MAX_BLOCKS = [undefined, // Level 0.
53   Infinity, Infinity, 2, 5, 5, 5, 5, 10, 7, 10][BlocklyGames.LEVEL];
54
55 // Crash type constants.
56 Robotmaze.CRASH_STOP = 1;
57 Robotmaze.CRASH_SPIN = 2;
58 Robotmaze.CRASH_FALL = 3;
59
60 Robotmaze.SKINS = [
61   // sprite: A 1029x51 set of 21 avatar images.
62   // tiles: A 250x200 set of 20 map images.
63   // marker: A 20x34 goal image.
64   // background: An optional 400x450 background image, or false.
65   // graph: Colour of optional grid lines, or false.
66   // look: Colour of sonar-like look icon.
67   // winSound: List of sounds (in various formats) to play when the player wins.
68   // crashSound: List of sounds (in various formats) for player crashes.
69   // crashType: Behaviour when player crashes (stop, spin, or fall).

```

```

70     {
71         sprite: 'robotmaze/pegman.png',
72         tiles: 'robotmaze/tiles_pegman.png',
73         marker: 'robotmaze/marker.png',
74         background: false,
75         graph: false,
76         look: '#000',
77         winSound: ['robotmaze/win.mp3', 'robotmaze/win.ogg'],
78         crashSound: ['robotmaze/fail_pegman.mp3', 'robotmaze/fail_pegman.ogg'],
79         crashType: Robotmaze.CRASH_STOP
80     },
81     {
82         sprite: 'robotmaze/astro.png',
83         tiles: 'robotmaze/tiles_astro.png',
84         marker: 'robotmaze/marker.png',
85         background: 'robotmaze/bg_astro.jpg',
86         // Coma star cluster, photo by George Hatfield, used with permission.
87         graph: false,
88         look: '#fff',
89         winSound: ['robotmaze/win.mp3', 'robotmaze/win.ogg'],
90         crashSound: ['robotmaze/fail_astro.mp3', 'robotmaze/fail_astro.ogg'],
91         crashType: Robotmaze.CRASH_SPIN
92     },
93     {
94         sprite: 'robotmaze/panda.png',
95         tiles: 'robotmaze/tiles_panda.png',
96         marker: 'robotmaze/marker.png',
97         background: 'robotmaze/bg_panda.jpg',
98         // Spring canopy, photo by Rupert Fleetingly, CC licensed for reuse.
99         graph: false,
100        look: '#000',
101        winSound: ['robotmaze/win.mp3', 'robotmaze/win.ogg'],
102        crashSound: ['robotmaze/fail_panda.mp3', 'robotmaze/fail_panda.ogg'],
103        crashType: Robotmaze.CRASH_FALL
104    }
105 ];
106 Robotmaze.SKIN_ID = BlocklyGames.getNumberParamFromUrl('skin', 0,
Robotmaze.SKINS.length);
107 Robotmaze.SKIN = Robotmaze.SKINS[Robotmaze.SKIN_ID];
108
109 /**
110  * Milliseconds between each animation frame.
111  */
112 Robotmaze.stepSpeed;
113
114 /**
115  * The types of squares in the robotmaze, which is represented
116  * as a 2D array of SquareType values.
117  * @enum {number}
118  */
119 Robotmaze.SquareType = {
120     WALL: 0,
121     OPEN: 1,
122     START: 2,
123     FINISH: 3
124 };
125
126 // The robotmaze square constants defined above are inlined here
127 // for ease of reading and writing the static robotmazes.
128 Robotmaze.map = [
129     // Level 0.
130     undefined,
131     // Level 1.
132     [[0, 0, 0, 0, 0, 0, 0, 0],
133      [0, 0, 0, 0, 0, 0, 0, 0],
134      [0, 0, 0, 0, 0, 0, 0, 0],
135      [0, 0, 0, 0, 0, 0, 0, 0],
136      [0, 0, 2, 1, 3, 0, 0, 0],
137      [0, 0, 0, 0, 0, 0, 0, 0],

```

```

138     [0, 0, 0, 0, 0, 0, 0, 0]],
139 // Level 2.
140 [[0, 0, 0, 0, 0, 0, 0, 0],
141  [0, 0, 0, 0, 0, 0, 0, 0],
142  [0, 0, 0, 0, 0, 0, 0, 0],
143  [0, 0, 0, 1, 3, 0, 0, 0],
144  [0, 0, 2, 1, 0, 0, 0, 0],
145  [0, 0, 0, 0, 0, 0, 0, 0],
146  [0, 0, 0, 0, 0, 0, 0, 0],
147  [0, 0, 0, 0, 0, 0, 0, 0]],
148 // Level 3.
149 [[0, 0, 0, 0, 0, 0, 0, 0],
150  [0, 0, 0, 0, 0, 0, 0, 0],
151  [0, 0, 0, 0, 0, 0, 0, 0],
152  [0, 0, 0, 0, 0, 0, 0, 0],
153  [0, 2, 1, 1, 1, 1, 3, 0],
154  [0, 0, 0, 0, 0, 0, 0, 0],
155  [0, 0, 0, 0, 0, 0, 0, 0],
156  [0, 0, 0, 0, 0, 0, 0, 0]],
157 // Level 4.
158 /**
159  * Note, the path continues past the start and the goal in both directions.
160  * This is intentionally done so users see the robotmaze is about getting from
161  * the start to the goal and not necessarily about moving over every part of
162  * the robotmaze, 'mowing the lawn' as Neil calls it.
163  */
164 [[0, 0, 0, 0, 0, 0, 0, 1],
165  [0, 0, 0, 0, 0, 0, 1, 1],
166  [0, 0, 0, 0, 0, 3, 1, 0],
167  [0, 0, 0, 0, 1, 1, 0, 0],
168  [0, 0, 0, 1, 1, 0, 0, 0],
169  [0, 0, 1, 1, 0, 0, 0, 0],
170  [0, 2, 1, 0, 0, 0, 0, 0],
171  [1, 1, 0, 0, 0, 0, 0, 0]],
172 // Level 5.
173 [[0, 0, 0, 0, 0, 0, 0, 0],
174  [0, 0, 0, 0, 0, 3, 0, 0],
175  [0, 0, 0, 0, 0, 1, 0, 0],
176  [0, 0, 0, 0, 0, 1, 0, 0],
177  [0, 0, 0, 0, 0, 1, 0, 0],
178  [0, 0, 0, 0, 0, 1, 0, 0],
179  [0, 0, 0, 2, 1, 1, 0, 0],
180  [0, 0, 0, 0, 0, 0, 0, 0]],
181 // Level 6.
182 [[0, 0, 0, 0, 0, 0, 0, 0],
183  [0, 0, 0, 0, 0, 0, 0, 0],
184  [0, 1, 1, 1, 1, 1, 0, 0],
185  [0, 1, 0, 0, 0, 1, 0, 0],
186  [0, 1, 1, 3, 0, 1, 0, 0],
187  [0, 0, 0, 0, 0, 1, 0, 0],
188  [0, 2, 1, 1, 1, 1, 0, 0],
189  [0, 0, 0, 0, 0, 0, 0, 0]],
190 // Level 7.
191 [[0, 0, 0, 0, 0, 0, 0, 0],
192  [0, 0, 0, 0, 0, 1, 1, 0],
193  [0, 2, 1, 1, 1, 1, 0, 0],
194  [0, 0, 0, 0, 0, 1, 1, 0],
195  [0, 1, 1, 3, 0, 1, 0, 0],
196  [0, 1, 0, 1, 0, 1, 0, 0],
197  [0, 1, 1, 1, 1, 1, 1, 0],
198  [0, 0, 0, 0, 0, 0, 0, 0]],
199 // Level 8.
200 [[0, 0, 0, 0, 0, 0, 0, 0],
201  [0, 0, 0, 0, 0, 0, 0, 0],
202  [0, 1, 1, 1, 1, 0, 0, 0],
203  [0, 1, 0, 0, 1, 1, 0, 0],
204  [0, 1, 1, 1, 0, 1, 0, 0],
205  [0, 0, 0, 1, 0, 1, 0, 0],
206  [0, 2, 1, 1, 0, 3, 0, 0]],

```

```

207     [0, 0, 0, 0, 0, 0, 0, 0, 0]],
208 // Level 9.
209 [[0, 0, 0, 0, 0, 0, 0, 0, 0],
210  [0, 1, 1, 1, 1, 1, 0, 0, 0],
211  [0, 0, 1, 0, 0, 0, 0, 0, 0],
212  [3, 1, 1, 1, 1, 1, 1, 0, 0],
213  [0, 1, 0, 1, 0, 1, 1, 0, 0],
214  [1, 1, 1, 1, 1, 0, 1, 0, 0],
215  [0, 1, 0, 1, 0, 2, 1, 0, 0],
216  [0, 0, 0, 0, 0, 0, 0, 0, 0]],
217 // Level 10.
218 [[0, 0, 0, 0, 0, 0, 0, 0, 0],
219  [0, 1, 1, 0, 3, 0, 1, 0, 0],
220  [0, 1, 1, 0, 1, 1, 1, 0, 0],
221  [0, 1, 0, 1, 0, 1, 0, 0, 0],
222  [0, 1, 1, 1, 1, 1, 1, 0, 0],
223  [0, 0, 0, 1, 0, 0, 1, 0, 0],
224  [0, 2, 1, 1, 1, 0, 1, 0, 0],
225  [0, 0, 0, 0, 0, 0, 0, 0, 0]]
226 ][BlocklyGames.LEVEL];
227
228 /**
229  * Measure robotmaze dimensions and set sizes.
230  * ROWS: Number of tiles down.
231  * COLS: Number of tiles across.
232  * SQUARE_SIZE: Pixel height and width of each robotmaze square (i.e. tile).
233  */
234 Robotmaze.ROWS = Robotmaze.map.length;
235 Robotmaze.COLS = Robotmaze.map[0].length;
236 Robotmaze.SQUARE_SIZE = 50;
237 Robotmaze.PEGMAN_HEIGHT = 52;
238 Robotmaze.PEGMAN_WIDTH = 49;
239
240 Robotmaze.MAZE_WIDTH = Robotmaze.SQUARE_SIZE * Robotmaze.COLS;
241 Robotmaze.MAZE_HEIGHT = Robotmaze.SQUARE_SIZE * Robotmaze.ROWS;
242 Robotmaze.PATH_WIDTH = Robotmaze.SQUARE_SIZE / 3;
243
244 /**
245  * Constants for cardinal directions. Subsequent code assumes these are
246  * in the range 0..3 and that opposites have an absolute difference of 2.
247  * @enum {number}
248  */
249 Robotmaze.DirectionType = {
250     NORTH: 0,
251     EAST: 1,
252     SOUTH: 2,
253     WEST: 3
254 };
255
256 /**
257  * Outcomes of running the user program.
258  */
259 Robotmaze.ResultType = {
260     UNSET: 0,
261     SUCCESS: 1,
262     FAILURE: -1,
263     TIMEOUT: 2,
264     ERROR: -2
265 };
266
267 /**
268  * Result of last execution.
269  */
270 Robotmaze.result = Robotmaze.ResultType.UNSET;
271
272 /**
273  * Starting direction.
274  */
275 Robotmaze.startDirection = Robotmaze.DirectionType.EAST;

```

```

276
277 /**
278  * PIDs of animation tasks currently executing.
279  */
280 Robotmaze.pidList = [];
281
282 // Map each possible shape to a sprite.
283 // Input: Binary string representing Centre/North/West/South/East squares.
284 // Output: [x, y] coordinates of each tile's sprite in tiles.png.
285 Robotmaze.tile_SHAPES = {
286   '10010': [4, 0], // Dead ends
287   '10001': [3, 3],
288   '11000': [0, 1],
289   '10100': [0, 2],
290   '11010': [4, 1], // Vertical
291   '10101': [3, 2], // Horizontal
292   '10110': [0, 0], // Elbows
293   '10011': [2, 0],
294   '11001': [4, 2],
295   '11100': [2, 3],
296   '11110': [1, 1], // Junctions
297   '10111': [1, 0],
298   '11011': [2, 1],
299   '11101': [1, 2],
300   '11111': [2, 2], // Cross
301   'null0': [4, 3], // Empty
302   'null1': [3, 0],
303   'null2': [3, 1],
304   'null3': [0, 3],
305   'null4': [1, 3]
306 };
307
308 /**
309  * Create and layout all the nodes for the path, scenery, Pegman, and goal.
310  */
311 Robotmaze.drawMap = function() {
312   var svg = document.getElementById('svgRobotmaze');
313   var scale = Math.max(Robotmaze.ROWS, Robotmaze.COLS) * Robotmaze.SQUARE_SIZE;
314   svg.setAttribute('viewBox', '0 0 ' + scale + ' ' + scale);
315
316   // Draw the outer square.
317   var square = document.createElementNS(Blockly.SVG_NS, 'rect');
318   square.setAttribute('width', Robotmaze.MAZE_WIDTH);
319   square.setAttribute('height', Robotmaze.MAZE_HEIGHT);
320   square.setAttribute('fill', '#F1EEE7');
321   square.setAttribute('stroke-width', 1);
322   square.setAttribute('stroke', '#CCB');
323   svg.appendChild(square);
324
325   if (Robotmaze.SKIN.background) {
326     var tile = document.createElementNS(Blockly.SVG_NS, 'image');
327     tile.setAttributeNS('http://www.w3.org/1999/xlink', 'xlink:href',
328       Robotmaze.SKIN.background);
329     tile.setAttribute('height', Robotmaze.MAZE_HEIGHT);
330     tile.setAttribute('width', Robotmaze.MAZE_WIDTH);
331     tile.setAttribute('x', 0);
332     tile.setAttribute('y', 0);
333     svg.appendChild(tile);
334   }
335
336   if (Robotmaze.SKIN.graph) {
337     // Draw the grid lines.
338     // The grid lines are offset so that the lines pass through the centre of
339     // each square. A half-pixel offset is also added to as standard SVG
340     // practice to avoid blurriness.
341     var offset = Robotmaze.SQUARE_SIZE / 2 + 0.5;
342     for (var k = 0; k < Robotmaze.ROWS; k++) {
343       var h_line = document.createElementNS(Blockly.SVG_NS, 'line');
344       h_line.setAttribute('y1', k * Robotmaze.SQUARE_SIZE + offset);

```

```

345     h_line.setAttribute('x2', Robotmaze.MAZE_WIDTH);
346     h_line.setAttribute('y2', k * Robotmaze.SQUARE_SIZE + offset);
347     h_line.setAttribute('stroke', Robotmaze.SKIN.graph);
348     h_line.setAttribute('stroke-width', 1);
349     svg.appendChild(h_line);
350 }
351 for (var k = 0; k < Robotmaze.COLS; k++) {
352     var v_line = document.createElementNS(Blockly.SVG_NS, 'line');
353     v_line.setAttribute('x1', k * Robotmaze.SQUARE_SIZE + offset);
354     v_line.setAttribute('x2', k * Robotmaze.SQUARE_SIZE + offset);
355     v_line.setAttribute('y2', Robotmaze.MAZE_HEIGHT);
356     v_line.setAttribute('stroke', Robotmaze.SKIN.graph);
357     v_line.setAttribute('stroke-width', 1);
358     svg.appendChild(v_line);
359 }
360 }
361
362 // Draw the tiles making up the robotmaze map.
363
364 // Return a value of '0' if the specified square is wall or out of bounds,
365 // '1' otherwise (empty, start, finish).
366 var normalize = function(x, y) {
367     if (x < 0 || x >= Robotmaze.COLS || y < 0 || y >= Robotmaze.ROWS) {
368         return '0';
369     }
370     return (Robotmaze.map[y][x] == Robotmaze.SquareType.WALL) ? '0' : '1';
371 };
372
373 // Compute and draw the tile for each square.
374 var tileId = 0;
375 for (var y = 0; y < Robotmaze.ROWS; y++) {
376     for (var x = 0; x < Robotmaze.COLS; x++) {
377         // Compute the tile shape.
378         var tileShape = normalize(x, y) +
379             normalize(x, y - 1) + // North.
380             normalize(x + 1, y) + // West.
381             normalize(x, y + 1) + // South.
382             normalize(x - 1, y); // East.
383
384         // Draw the tile.
385         if (!Robotmaze.tile_SHAPES[tileShape]) {
386             // Empty square. Use null0 for large areas, with null1-4 for borders.
387             // Add some randomness to avoid large empty spaces.
388             if (tileShape == '00000' && Math.random() > 0.3) {
389                 tileShape = 'null0';
390             } else {
391                 tileShape = 'null' + Math.floor(1 + Math.random() * 4);
392             }
393         }
394         var left = Robotmaze.tile_SHAPES[tileShape][0];
395         var top = Robotmaze.tile_SHAPES[tileShape][1];
396         // Tile's clipPath element.
397         var tileClip = document.createElementNS(Blockly.SVG_NS, 'clipPath');
398         tileClip.setAttribute('id', 'tileClipPath' + tileId);
399         var clipRect = document.createElementNS(Blockly.SVG_NS, 'rect');
400         clipRect.setAttribute('width', Robotmaze.SQUARE_SIZE);
401         clipRect.setAttribute('height', Robotmaze.SQUARE_SIZE);
402
403         clipRect.setAttribute('x', x * Robotmaze.SQUARE_SIZE);
404         clipRect.setAttribute('y', y * Robotmaze.SQUARE_SIZE);
405
406         tileClip.appendChild(clipRect);
407         svg.appendChild(tileClip);
408         // Tile sprite.
409         var tile = document.createElementNS(Blockly.SVG_NS, 'image');
410         tile.setAttributeNS('http://www.w3.org/1999/xlink', 'xlink:href',
411             Robotmaze.SKIN.tiles);
412         // Position the tile sprite relative to the clipRect.
413         tile.setAttribute('height', Robotmaze.SQUARE_SIZE * 4);

```

```

414     tile.setAttribute('width', Robotmaze.SQUARE_SIZE * 5);
415     tile.setAttribute('clip-path', 'url(#tileClipPath' + tileId + ')');
416     tile.setAttribute('x', (x - left) * Robotmaze.SQUARE_SIZE);
417     tile.setAttribute('y', (y - top) * Robotmaze.SQUARE_SIZE);
418     svg.appendChild(tile);
419     tileId++;
420 }
421 }
422
423 // Add finish marker.
424 var finishMarker = document.createElementNS(Blockly.SVG_NS, 'image');
425 finishMarker.setAttribute('id', 'finish');
426 finishMarker.setAttributeNS('http://www.w3.org/1999/xlink', 'xlink:href',
427     Robotmaze.SKIN.marker);
428 finishMarker.setAttribute('height', 34);
429 finishMarker.setAttribute('width', 20);
430 svg.appendChild(finishMarker);
431
432 // Pegman's clipPath element, whose (x, y) is reset by Robotmaze.displayPegman
433 var pegmanClip = document.createElementNS(Blockly.SVG_NS, 'clipPath');
434 pegmanClip.setAttribute('id', 'pegmanClipPath');
435 var clipRect = document.createElementNS(Blockly.SVG_NS, 'rect');
436 clipRect.setAttribute('id', 'clipRect');
437 clipRect.setAttribute('width', Robotmaze.PEGMAN_WIDTH);
438 clipRect.setAttribute('height', Robotmaze.PEGMAN_HEIGHT);
439 pegmanClip.appendChild(clipRect);
440 svg.appendChild(pegmanClip);
441
442 // Add Pegman.
443 var pegmanIcon = document.createElementNS(Blockly.SVG_NS, 'image');
444 pegmanIcon.setAttribute('id', 'pegman');
445 pegmanIcon.setAttributeNS('http://www.w3.org/1999/xlink', 'xlink:href',
446     Robotmaze.SKIN.sprite);
447 pegmanIcon.setAttribute('height', Robotmaze.PEGMAN_HEIGHT);
448 pegmanIcon.setAttribute('width', Robotmaze.PEGMAN_WIDTH * 21); // 49 * 21 = 1029
449 pegmanIcon.setAttribute('clip-path', 'url(#pegmanClipPath)');
450 svg.appendChild(pegmanIcon);
451 };
452
453 /**
454  * Initialize Blockly and the robotmaze. Called on page load.
455  */
456 Robotmaze.init = function() {
457     // Render the Soy template.
458     document.body.innerHTML = Robotmaze.soy.start({}, null,
459         {lang: BlocklyGames.LANG,
460         level: BlocklyGames.LEVEL,
461         maxLevel: BlocklyGames.MAX_LEVEL,
462         skin: Robotmaze.SKIN_ID,
463         html: BlocklyGames.IS_HTML});
464
465     BlocklyInterface.init();
466
467     // Setup the Pegman menu.
468     var pegmanImg = document.querySelector('#pegmanButton>img');
469     pegmanImg.style.backgroundImage = 'url(' + Robotmaze.SKIN.sprite + ')';
470     var pegmanMenu = document.getElementById('pegmanMenu');
471     var handlerFactory = function(n) {
472         return function() {
473             Robotmaze.changePegman(n);
474         };
475     };
476     for (var i = 0; i < Robotmaze.SKINS.length; i++) {
477         if (i == Robotmaze.SKIN_ID) {
478             continue;
479         }
480         var div = document.createElement('div');
481         var img = document.createElement('img');
482         img.src = 'common/1x1.gif';

```

```

483     img.style.backgroundImage = 'url(' + Robotmaze.SKINS[i].sprite + ')';
484     div.appendChild(img);
485     pegmanMenu.appendChild(div);
486     Blockly.bindEvent_(div, 'mousedown', null, handlerFactory(i));
487 }
488 Blockly.bindEvent_(window, 'resize', null, Robotmaze.hidePegmanMenu);
489 var pegmanButton = document.getElementById('pegmanButton');
490 Blockly.bindEvent_(pegmanButton, 'mousedown', null, Robotmaze.showPegmanMenu);
491 var pegmanButtonArrow = document.getElementById('pegmanButtonArrow');
492 var arrow = document.createTextNode(Blockly.FieldDropdown.ARROW_CHAR);
493 pegmanButtonArrow.appendChild(arrow);
494
495 var rtl = BlocklyGames.isRtl();
496 var blocklyDiv = document.getElementById('blockly');
497 var visualization = document.getElementById('visualization');
498 var onresize = function(e) {
499     var top = visualization.offsetTop;
500     blocklyDiv.style.top = Math.max(10, top - window.pageYOffset) + 'px';
501     blocklyDiv.style.left = rtl ? '10px' : '420px';
502     blocklyDiv.style.width = (window.innerWidth - 440) + 'px';
503 };
504 window.addEventListener('scroll', function() {
505     onresize(null);
506     Blockly.svgResize(BlocklyGames.workspace);
507 });
508 window.addEventListener('resize', onresize);
509 onresize(null);
510
511 var toolbox = document.getElementById('toolbox');
512 // Scale the workspace so level 1 = 1.3, and level 10 = 1.0.
513 var scale = 1 + (1 - (BlocklyGames.LEVEL / BlocklyGames.MAX_LEVEL)) / 3;
514 BlocklyGames.workspace = Blockly.inject('blockly',
515     {
516         'media': 'third-party/blockly/media/',
517         'maxBlocks': Robotmaze.MAX_BLOCKS,
518         'rtl': rtl,
519         'toolbox': toolbox,
520         'trashcan': true,
521         'zoom': {'startScale': scale}
522     });
523 BlocklyGames.workspace.getAudioManager().load(Robotmaze.SKIN.winSound, 'win');
524 BlocklyGames.workspace.getAudioManager().load(Robotmaze.SKIN.crashSound, 'fail');
525 // Not really needed, there are no user-defined functions or variables.
526 Blockly.JavaScript.addReservedWords('moveForward,moveBackward,' +
527     'turnRight,turnLeft,isPathForward,isPathRight,isPathBackward,isPathLeft');
528
529 Robotmaze.drawMap();
530
531 var defaultXml =
532     '<xml>' +
533     '  <block movable="' + (BlocklyGames.LEVEL != 1) + '" ' +
534     'type="robotmaze_moveForward" x="70" y="70"></block>' +
535     '</xml>';
536 BlocklyInterface.loadBlocks(defaultXml, false);
537
538 // Locate the start and finish squares.
539 for (var y = 0; y < Robotmaze.ROWS; y++) {
540     for (var x = 0; x < Robotmaze.COLS; x++) {
541         if (Robotmaze.map[y][x] == Robotmaze.SquareType.START) {
542             Robotmaze.start_ = {x: x, y: y};
543         } else if (Robotmaze.map[y][x] == Robotmaze.SquareType.FINISH) {
544             Robotmaze.finish_ = {x: x, y: y};
545         }
546     }
547 }
548
549 Robotmaze.reset(true);
550 BlocklyGames.workspace.addChangeListener(function() {Robotmaze.updateCapacity();});
551
552 document.body.addEventListener('mousemove', Robotmaze.updatePegSpin_, true);

```



```

552 BlocklyGames.bindClick('runButton', Robotmaze.runButtonClick);
553 BlocklyGames.bindClick('resetButton', Robotmaze.resetButtonClick);
554
555 if (BlocklyGames.LEVEL == 1) {
556     // Make connecting blocks easier for beginners.
557     Blockly.SNAP_RADIUS *= 2;
558     Blockly.CONNECTING_SNAP_RADIUS = Blockly.SNAP_RADIUS;
559 }
560 if (BlocklyGames.LEVEL == 10) {
561     if (!BlocklyGames.loadFromLocalStorage(BlocklyGames.NAME,
562                                             BlocklyGames.LEVEL)) {
563         // Level 10 gets an introductory modal dialog.
564         // Skip the dialog if the user has already won.
565         var content = document.getElementById('dialogHelpWallFollow');
566         var style = {
567             'width': '30%',
568             'left': '35%',
569             'top': '12em'
570         };
571         BlocklyDialogs.showDialog(content, null, false, true, style,
572                                   BlocklyDialogs.stopDialogKeyDown);
573         BlocklyDialogs.startDialogKeyDown();
574         setTimeout(BlocklyDialogs.abortOffer, 5 * 60 * 1000);
575     }
576 } else {
577     // All other levels get interactive help. But wait 5 seconds for the
578     // user to think a bit before they are told what to do.
579     setTimeout(function() {
580         BlocklyGames.workspace.addChangeListener(Robotmaze.levelHelp);
581         Robotmaze.levelHelp();
582     }, 5000);
583 }
584
585 // Add the spinning Pegman icon to the done dialog.
586 // 
587 var buttonDiv = document.getElementById('dialogDoneButtons');
588 var pegSpin = document.createElement('img');
589 pegSpin.id = 'pegSpin';
590 pegSpin.src = 'common/1x1.gif';
591 pegSpin.style.backgroundImage = 'url(' + Robotmaze.SKIN.sprite + ')';
592 buttonDiv.parentNode.insertBefore(pegSpin, buttonDiv);
593
594 // Lazy-load the JavaScript interpreter.
595 setTimeout(BlocklyInterface.importInterpreter, 1);
596 // Lazy-load the syntax-highlighting.
597 setTimeout(BlocklyInterface.importPrettify, 1);
598 };
599
600 /**
601  * When the workspace changes, update the help as needed.
602  * @param {Blockly.Events.Abstract=} opt_event Custom data for event.
603  */
604 Robotmaze.levelHelp = function(opt_event) {
605     if (opt_event && opt_event.type == Blockly.Events.UI) {
606         // Just a change to highlighting or somesuch.
607         return;
608     } else if (BlocklyGames.workspace.isDragging()) {
609         // Don't change helps during drags.
610         return;
611     } else if (Robotmaze.result == Robotmaze.ResultType.SUCCESS ||
612               BlocklyGames.loadFromLocalStorage(BlocklyGames.NAME,
613                                                  BlocklyGames.LEVEL)) {
614         // The user has already won. They are just playing around.
615         return;
616     }
617     var rtl = BlocklyGames.isRtl();
618     var userBlocks = Blockly.Xml.domToText(
619         Blockly.Xml.workspaceToDom(BlocklyGames.workspace));
620     var toolbar = BlocklyGames.workspace.flyout_.workspace_.getTopBlocks(true);

```

```

621 var content = null;
622 var origin = null;
623 var style = null;
624 if (BlocklyGames.LEVEL == 1) {
625     if (BlocklyGames.workspace.getAllBlocks().length < 2) {
626         content = document.getElementById('dialogHelpStack');
627         style = {'width': '370px', 'top': '130px'};
628         style[rtl ? 'right' : 'left'] = '215px';
629         origin = toolbar[0].getSvgRoot();
630     } else {
631         var topBlocks = BlocklyGames.workspace.getTopBlocks(true);
632         if (topBlocks.length > 1) {
633             var xml = [
634                 '<xml>',
635                 '<block type="robotmaze_moveForward" x="10" y="10">',
636                 '<next>',
637                 '<block type="robotmaze_moveForward"></block>',
638                 '</next>',
639                 '</block>',
640                 '</xml>'];
641             BlocklyInterface.injectReadOnly('sampleOneTopBlock', xml);
642             content = document.getElementById('dialogHelpOneTopBlock');
643             style = {'width': '360px', 'top': '120px'};
644             style[rtl ? 'right' : 'left'] = '225px';
645             origin = topBlocks[0].getSvgRoot();
646         } else if (Robotmaze.result == Robotmaze.ResultType.UNSET) {
647             // Show run help dialog.
648             content = document.getElementById('dialogHelpRun');
649             style = {'width': '360px', 'top': '410px'};
650             style[rtl ? 'right' : 'left'] = '400px';
651             origin = document.getElementById('runButton');
652         }
653     }
654 } else if (BlocklyGames.LEVEL == 2) {
655     if (Robotmaze.result != Robotmaze.ResultType.UNSET &&
656         document.getElementById('runButton').style.display == 'none') {
657         content = document.getElementById('dialogHelpReset');
658         style = {'width': '360px', 'top': '410px'};
659         style[rtl ? 'right' : 'left'] = '400px';
660         origin = document.getElementById('resetButton');
661     }
662 } else if (BlocklyGames.LEVEL == 3) {
663     if (userBlocks.indexOf('robotmaze_forever') == -1) {
664         if (BlocklyGames.workspace.remainingCapacity() == 0) {
665             content = document.getElementById('dialogHelpCapacity');
666             style = {'width': '430px', 'top': '310px'};
667             style[rtl ? 'right' : 'left'] = '50px';
668             origin = document.getElementById('capacityBubble');
669         } else {
670             content = document.getElementById('dialogHelpRepeat');
671             style = {'width': '360px', 'top': '360px'};
672             style[rtl ? 'right' : 'left'] = '425px';
673             origin = toolbar[3].getSvgRoot();
674         }
675     }
676 } else if (BlocklyGames.LEVEL == 4) {
677     if (BlocklyGames.workspace.remainingCapacity() == 0 &&
678         (userBlocks.indexOf('robotmaze_forever') == -1 ||
679         BlocklyGames.workspace.getTopBlocks(false).length > 1)) {
680         content = document.getElementById('dialogHelpCapacity');
681         style = {'width': '430px', 'top': '310px'};
682         style[rtl ? 'right' : 'left'] = '50px';
683         origin = document.getElementById('capacityBubble');
684     } else {
685         var showHelp = true;
686         // Only show help if there is not a loop with two nested blocks.
687         var blocks = BlocklyGames.workspace.getAllBlocks();
688         for (var i = 0; i < blocks.length; i++) {
689             var block = blocks[i];

```

```

690         if (block.type !== 'robotmaze_forever') {
691             continue;
692         }
693         var j = 0;
694         while (block) {
695             var kids = block.getChildren();
696             block = kids.length ? kids[0] : null;
697             j++;
698         }
699         if (j > 2) {
700             showHelp = false;
701             break;
702         }
703     }
704     if (showHelp) {
705         content = document.getElementById('dialogHelpRepeatMany');
706         style = {'width': '360px', 'top': '360px'};
707         style[rtl ? 'right' : 'left'] = '425px';
708         origin = toolbar[3].getSvgRoot();
709     }
710 }
711 } else if (BlocklyGames.LEVEL == 5) {
712     if (Robotmaze.SKIN_ID == 0 && !Robotmaze.showPegmanMenu.activatedOnce) {
713         content = document.getElementById('dialogHelpSkins');
714         style = {'width': '360px', 'top': '60px'};
715         style[rtl ? 'left' : 'right'] = '20px';
716         origin = document.getElementById('pegmanButton');
717     }
718 } else if (BlocklyGames.LEVEL == 6) {
719     if (userBlocks.indexOf('robotmaze_if') == -1) {
720         content = document.getElementById('dialogHelpIf');
721         style = {'width': '360px', 'top': '430px'};
722         style[rtl ? 'right' : 'left'] = '425px';
723         origin = toolbar[4].getSvgRoot();
724     }
725 } else if (BlocklyGames.LEVEL == 7) {
726     if (!Robotmaze.levelHelp.initialized7_) {
727         // Create fake dropdown.
728         var span = document.createElement('span');
729         span.className = 'helpMenuFake';
730         var options =
731             [BlocklyGames.getMsg('Robotmaze_pathAhead'),
732              BlocklyGames.getMsg('Robotmaze_pathLeft'),
733              BlocklyGames.getMsg('Robotmaze_pathRight')];
734         var prefix = Blockly.utils.commonWordPrefix(options);
735         var suffix = Blockly.utils.commonWordSuffix(options);
736         if (suffix) {
737             var option = options[0].slice(prefix, -suffix);
738         } else {
739             var option = options[0].substring(prefix);
740         }
741         // Add dropdown arrow: "option ▼" (LTR) or "▼ אופציה" (RTL)
742         span.textContent = option + ' ' + Blockly.FieldDropdown.ARROW_CHAR;
743         // Inject fake dropdown into message.
744         var container = document.getElementById('helpMenuText');
745         var msg = container.textContent;
746         container.textContent = '';
747         var parts = msg.split(/%\d/);
748         for (var i = 0; i < parts.length; i++) {
749             container.appendChild(document.createTextNode(parts[i]));
750             if (i !== parts.length - 1) {
751                 container.appendChild(span.cloneNode(true));
752             }
753         }
754         Robotmaze.levelHelp.initialized7_ = true;
755     }
756     // The hint says to change from 'ahead', but keep the hint visible
757     // until the user chooses 'right'.
758     if (userBlocks.indexOf('isPathRight') == -1) {

```

```

759     content = document.getElementById('dialogHelpMenu');
760     style = {'width': '360px', 'top': '430px'};
761     style[rtl ? 'right' : 'left'] = '425px';
762     origin = toolbar[4].getSvgRoot();
763 }
764 } else if (BlocklyGames.LEVEL == 9) {
765     if (userBlocks.indexOf('robotmaze_ifElse') == -1) {
766         content = document.getElementById('dialogHelpIfElse');
767         style = {'width': '360px', 'top': '305px'};
768         style[rtl ? 'right' : 'left'] = '425px';
769         origin = toolbar[5].getSvgRoot();
770     }
771 }
772 if (content) {
773     if (content.parentNode != document.getElementById('dialog')) {
774         BlocklyDialogs.showDialog(content, origin, true, false, style, null);
775     }
776 } else {
777     BlocklyDialogs.hideDialog(false);
778 }
779 };
780
781 /**
782  * Reload with a different Pegman skin.
783  * @param {number} newSkin ID of new skin.
784  */
785 Robotmaze.changePegman = function(newSkin) {
786     Robotmaze.saveToStorage();
787     window.location = window.location.protocol + '//' +
788         window.location.host + window.location.pathname +
789         '?lang=' + BlocklyGames.LANG + '&level=' + BlocklyGames.LEVEL +
790         '&skin=' + newSkin;
791 };
792
793 /**
794  * Save the blocks for a one-time reload.
795  */
796 Robotmaze.saveToStorage = function() {
797     // MSIE 11 does not support sessionStorage on file:// URLs.
798     if (typeof Blockly != undefined && window.sessionStorage) {
799         var xml = Blockly.Xml.workspaceToDom(BlocklyGames.workspace);
800         var text = Blockly.Xml.domToText(xml);
801         window.sessionStorage.loadOnceBlocks = text;
802     }
803 };
804
805 /**
806  * Display the Pegman skin-change menu.
807  * @param {!Event} e Mouse, touch, or resize event.
808  */
809 Robotmaze.showPegmanMenu = function(e) {
810     var menu = document.getElementById('pegmanMenu');
811     if (menu.style.display == 'block') {
812         // Menu is already open. Close it.
813         Robotmaze.hidePegmanMenu(e);
814         return;
815     }
816     // Prevent double-clicks or double-taps.
817     if (BlocklyInterface.eventSpam(e)) {
818         return;
819     }
820     var button = document.getElementById('pegmanButton');
821     button.classList.add('buttonHover');
822     menu.style.top = (button.offsetTop + button.offsetHeight) + 'px';
823     menu.style.left = button.offsetLeft + 'px';
824     menu.style.display = 'block';
825     Robotmaze.pegmanMenuMouse_ =
826         Blockly.bindEvent_(document.body, 'mousedown', null, Robotmaze.hidePegmanMenu);
827     // Close the skin-changing hint if open.

```

```

828     var hint = document.getElementById('dialogHelpSkins');
829     if (hint && hint.className !== 'dialogHiddenContent') {
830         BlocklyDialogs.hideDialog(false);
831     }
832     Robotmaze.showPegmanMenu.activatedOnce = true;
833 };
834
835 /**
836  * Hide the Pegman skin-change menu.
837  * @param {!Event} e Mouse, touch, or resize event.
838  */
839 Robotmaze.hidePegmanMenu = function(e) {
840     // Prevent double-clicks or double-taps.
841     if (BlocklyInterface.eventSpam(e)) {
842         return;
843     }
844     document.getElementById('pegmanMenu').style.display = 'none';
845     document.getElementById('pegmanButton').classList.remove('buttonHover');
846     if (Robotmaze.pegmanMenuMouse_) {
847         Blockly.unbindEvent_(Robotmaze.pegmanMenuMouse_);
848         delete Robotmaze.pegmanMenuMouse_;
849     }
850 };
851
852 /**
853  * Reset the robotmaze to the start position and kill any pending animation tasks.
854  * @param {boolean} first True if an opening animation is to be played.
855  */
856 Robotmaze.reset = function(first) {
857     // Kill all tasks.
858     for (var i = 0; i < Robotmaze.pidList.length; i++) {
859         window.clearTimeout(Robotmaze.pidList[i]);
860     }
861     Robotmaze.pidList = [];
862
863     // Move Pegman into position.
864     Robotmaze.pegmanX = Robotmaze.start_.x;
865     Robotmaze.pegmanY = Robotmaze.start_.y;
866
867     if (first) {
868         Robotmaze.pegmanD = Robotmaze.startDirection + 1;
869         Robotmaze.scheduleFinish(false);
870         Robotmaze.pidList.push(setTimeout(function() {
871             Robotmaze.stepSpeed = 100;
872             Robotmaze.schedule([Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4],
873                 [Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4 - 4]);
874             Robotmaze.pegmanD++;
875             }, Robotmaze.stepSpeed * 5));
876     } else {
877         Robotmaze.pegmanD = Robotmaze.startDirection;
878         Robotmaze.displayPegman(Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4);
879     }
880
881     // Move the finish icon into position.
882     var finishIcon = document.getElementById('finish');
883     finishIcon.setAttribute('x', Robotmaze.SQUARE_SIZE * (Robotmaze.finish_.x + 0.5) -
884         finishIcon.getAttribute('width') / 2);
885     finishIcon.setAttribute('y', Robotmaze.SQUARE_SIZE * (Robotmaze.finish_.y + 0.6) -
886         finishIcon.getAttribute('height'));
887
888     // Make 'look' icon invisible and promote to top.
889     var lookIcon = document.getElementById('look');
890     lookIcon.style.display = 'none';
891     lookIcon.parentNode.appendChild(lookIcon);
892     var paths = lookIcon.getElementsByTagName('path');
893     for (var i = 0, path; (path = paths[i]); i++) {
894         path.setAttribute('stroke', Robotmaze.SKIN.look);
895     }
896 };

```

```

897
898 /**
899  * Click the run button.  Start the program.
900  * @param {!Event} e Mouse or touch event.
901  */
902 Robotmaze.runButtonClick = function(e) {
903     // Prevent double-clicks or double-taps.
904     if (BlocklyInterface.eventSpam(e)) {
905         return;
906     }
907     BlocklyDialogs.hideDialog(false);
908     // Only allow a single top block on level 1.
909     if (BlocklyGames.LEVEL == 1 &&
910         BlocklyGames.workspace.getTopBlocks(false).length > 1 &&
911         Robotmaze.result != Robotmaze.ResultType.SUCCESS &&
912         !BlocklyGames.loadFromLocalStorage(BlocklyGames.NAME,
913                                             BlocklyGames.LEVEL)) {
914         Robotmaze.levelHelp();
915         return;
916     }
917     var runButton = document.getElementById('runButton');
918     var resetButton = document.getElementById('resetButton');
919     // Ensure that Reset button is at least as wide as Run button.
920     if (!resetButton.style.minWidth) {
921         resetButton.style.minWidth = runButton.offsetWidth + 'px';
922     }
923     runButton.style.display = 'none';
924     resetButton.style.display = 'inline';
925     Robotmaze.reset(false);
926     Robotmaze.execute();
927 };
928
929 /**
930  * Updates the document's 'capacity' element with a message
931  * indicating how many more blocks are permitted.  The capacity
932  * is retrieved from BlocklyGames.workspace.remainingCapacity().
933  */
934 Robotmaze.updateCapacity = function() {
935     var cap = BlocklyGames.workspace.remainingCapacity();
936     var p = document.getElementById('capacity');
937     if (cap == Infinity) {
938         p.style.display = 'none';
939     } else {
940         p.style.display = 'inline';
941         p.innerHTML = '';
942         cap = Number(cap);
943         var capSpan = document.createElement('span');
944         capSpan.className = 'capacityNumber';
945         capSpan.appendChild(document.createTextNode(cap));
946         if (cap == 0) {
947             var msg = BlocklyGames.getMsg('Robotmaze_capacity0');
948         } else if (cap == 1) {
949             var msg = BlocklyGames.getMsg('Robotmaze_capacity1');
950         } else {
951             var msg = BlocklyGames.getMsg('Robotmaze_capacity2');
952         }
953         var parts = msg.split(/%\d/);
954         for (var i = 0; i < parts.length; i++) {
955             p.appendChild(document.createTextNode(parts[i]));
956             if (i != parts.length - 1) {
957                 p.appendChild(capSpan.cloneNode(true));
958             }
959         }
960     }
961 };
962
963 /**
964  * Click the reset button.  Reset the robotmaze.
965  * @param {!Event} e Mouse or touch event.

```

```

966 */
967 Robotmaze.resetButtonClick = function(e) {
968     // Prevent double-clicks or double-taps.
969     if (BlocklyInterface.eventSpam(e)) {
970         return;
971     }
972     var runButton = document.getElementById('runButton');
973     runButton.style.display = 'inline';
974     document.getElementById('resetButton').style.display = 'none';
975     BlocklyGames.workspace.highlightBlock(null);
976     Robotmaze.reset(false);
977     Robotmaze.levelHelp();
978 };
979
980 /**
981  * Inject the Robotmaze API into a JavaScript interpreter.
982  * @param {!Interpreter} interpreter The JS Interpreter.
983  * @param {!Interpreter.Object} scope Global scope.
984  */
985 Robotmaze.initInterpreter = function(interpreter, scope) {
986     // API
987     var wrapper;
988     wrapper = function(id) {
989         Robotmaze.move(0, id);
990     };
991     interpreter.setProperty(scope, 'moveForward',
992         interpreter.createNativeFunction(wrapper));
993     wrapper = function(id) {
994         Robotmaze.move(2, id);
995     };
996     interpreter.setProperty(scope, 'moveBackward',
997         interpreter.createNativeFunction(wrapper));
998     wrapper = function(id) {
999         Robotmaze.turn(0, id);
1000     };
1001     interpreter.setProperty(scope, 'turnLeft',
1002         interpreter.createNativeFunction(wrapper));
1003     wrapper = function(id) {
1004         Robotmaze.turn(1, id);
1005     };
1006     interpreter.setProperty(scope, 'turnRight',
1007         interpreter.createNativeFunction(wrapper));
1008     wrapper = function(id) {
1009         return Robotmaze.isPath(0, id);
1010     };
1011     interpreter.setProperty(scope, 'isPathForward',
1012         interpreter.createNativeFunction(wrapper));
1013     wrapper = function(id) {
1014         return Robotmaze.isPath(1, id);
1015     };
1016     interpreter.setProperty(scope, 'isPathRight',
1017         interpreter.createNativeFunction(wrapper));
1018     wrapper = function(id) {
1019         return Robotmaze.isPath(2, id);
1020     };
1021     interpreter.setProperty(scope, 'isPathBackward',
1022         interpreter.createNativeFunction(wrapper));
1023     wrapper = function(id) {
1024         return Robotmaze.isPath(3, id);
1025     };
1026     interpreter.setProperty(scope, 'isPathLeft',
1027         interpreter.createNativeFunction(wrapper));
1028     wrapper = function() {
1029         return Robotmaze.notDone();
1030     };
1031     interpreter.setProperty(scope, 'notDone',
1032         interpreter.createNativeFunction(wrapper));
1033 };
1034

```



```

1035  /**
1036  * Execute the user's code.  Heaven help us...
1037  */
1038  Robotmaze.execute = function() {
1039      if (!('Interpreter' in window)) {
1040          // Interpreter lazy loads and hasn't arrived yet.  Try again later.
1041          setTimeout(Robotmaze.execute, 250);
1042          return;
1043      }
1044
1045      Robotmaze.log = [];
1046      Blockly.selected && Blockly.selected.unselect();
1047      var code = Blockly.JavaScript.workspaceToCode(BlocklyGames.workspace);
1048      Robotmaze.result = Robotmaze.ResultType.UNSET;
1049      var interpreter = new Interpreter(code, Robotmaze.initInterpreter);
1050
1051      // Try running the user's code.  There are four possible outcomes:
1052      // 1. If pegman reaches the finish [SUCCESS], true is thrown.
1053      // 2. If the program is terminated due to running too long [TIMEOUT],
1054      //    false is thrown.
1055      // 3. If another error occurs [ERROR], that error is thrown.
1056      // 4. If the program ended normally but without solving the robotmaze [FAILURE],
1057      //    no error or exception is thrown.
1058      try {
1059          var ticks = 10000; // 10k ticks runs Pegman for about 8 minutes.
1060          while (interpreter.step()) {
1061              if (ticks-- == 0) {
1062                  throw Infinity;
1063              }
1064          }
1065          Robotmaze.result = Robotmaze.notDone() ?
1066              Robotmaze.ResultType.FAILURE : Robotmaze.ResultType.SUCCESS;
1067      } catch (e) {
1068          // A boolean is thrown for normal termination.
1069          // Abnormal termination is a user error.
1070          if (e === Infinity) {
1071              Robotmaze.result = Robotmaze.ResultType.TIMEOUT;
1072          } else if (e === false) {
1073              Robotmaze.result = Robotmaze.ResultType.ERROR;
1074          } else {
1075              // Syntax error, can't happen.
1076              Robotmaze.result = Robotmaze.ResultType.ERROR;
1077              alert(e);
1078          }
1079      }
1080
1081      // Fast animation if execution is successful.  Slow otherwise.
1082      if (Robotmaze.result == Robotmaze.ResultType.SUCCESS) {
1083          Robotmaze.stepSpeed = 100;
1084          Robotmaze.log.push(['finish', null]);
1085      } else {
1086          Robotmaze.stepSpeed = 150;
1087      }
1088
1089      // Robotmaze.log now contains a transcript of all the user's actions.
1090      // Reset the robotmaze and animate the transcript.
1091      Robotmaze.reset(false);
1092      Robotmaze.pidList.push(setTimeout(Robotmaze.animate, 100));
1093  };
1094
1095  /**
1096  * Iterate through the recorded path and animate pegman's actions.
1097  */
1098  Robotmaze.animate = function() {
1099      var action = Robotmaze.log.shift();
1100      if (!action) {
1101          BlocklyInterface.highlight(null);
1102          Robotmaze.levelHelp();
1103          return;

```



```

1104     }
1105     BlocklyInterface.highlight(action[1]);
1106
1107     switch (action[0]) {
1108         case 'north':
1109             Robotmaze.schedule([Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4],
1110                 [Robotmaze.pegmanX, Robotmaze.pegmanY - 1, Robotmaze.pegmanD * 4]);
1111             Robotmaze.pegmanY--;
1112             break;
1113         case 'east':
1114             Robotmaze.schedule([Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4],
1115                 [Robotmaze.pegmanX + 1, Robotmaze.pegmanY, Robotmaze.pegmanD * 4]);
1116             Robotmaze.pegmanX++;
1117             break;
1118         case 'south':
1119             Robotmaze.schedule([Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4],
1120                 [Robotmaze.pegmanX, Robotmaze.pegmanY + 1, Robotmaze.pegmanD * 4]);
1121             Robotmaze.pegmanY++;
1122             break;
1123         case 'west':
1124             Robotmaze.schedule([Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4],
1125                 [Robotmaze.pegmanX - 1, Robotmaze.pegmanY, Robotmaze.pegmanD * 4]);
1126             Robotmaze.pegmanX--;
1127             break;
1128         case 'look_north':
1129             Robotmaze.scheduleLook(Robotmaze.DirectionType.NORTH);
1130             break;
1131         case 'look_east':
1132             Robotmaze.scheduleLook(Robotmaze.DirectionType.EAST);
1133             break;
1134         case 'look_south':
1135             Robotmaze.scheduleLook(Robotmaze.DirectionType.SOUTH);
1136             break;
1137         case 'look_west':
1138             Robotmaze.scheduleLook(Robotmaze.DirectionType.WEST);
1139             break;
1140         case 'fail_forward':
1141             Robotmaze.scheduleFail(true);
1142             break;
1143         case 'fail_backward':
1144             Robotmaze.scheduleFail(false);
1145             break;
1146         case 'left':
1147             Robotmaze.schedule([Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4],
1148                 [Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4 - 4]);
1149             Robotmaze.pegmanD = Robotmaze.constrainDirection4(Robotmaze.pegmanD - 1);
1150             break;
1151         case 'right':
1152             Robotmaze.schedule([Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4],
1153                 [Robotmaze.pegmanX, Robotmaze.pegmanY, Robotmaze.pegmanD * 4 + 4]);
1154             Robotmaze.pegmanD = Robotmaze.constrainDirection4(Robotmaze.pegmanD + 1);
1155             break;
1156         case 'finish':
1157             Robotmaze.scheduleFinish(true);
1158             BlocklyInterface.saveToLocalStorage();
1159             setTimeout(BlocklyDialogs.congratulations, 1000);
1160     }
1161
1162     Robotmaze.pidList.push(setTimeout(Robotmaze.animate, Robotmaze.stepSpeed * 5));
1163 };
1164
1165 /**
1166  * Point the congratulations Pegman to face the mouse.
1167  * @param {Event} e Mouse move event.
1168  * @private
1169  */
1170 Robotmaze.updatePegSpin_ = function(e) {
1171     if (document.getElementById('dialogDone').className ==
1172         'dialogHiddenContent') {

```

```

1173     return;
1174 }
1175 var pegSpin = document.getElementById('pegSpin');
1176 var bBox = BlocklyDialogs.getBBox_(pegSpin);
1177 var x = bBox.x + bBox.width / 2 - window.pageXOffset;
1178 var y = bBox.y + bBox.height / 2 - window.pageYOffset;
1179 var dx = e.clientX - x;
1180 var dy = e.clientY - y;
1181 var angle = Math.atan(dy / dx);
1182 // Convert from radians to degrees because I suck at math.
1183 angle = angle / Math.PI * 180;
1184 // 0: North, 90: East, 180: South, 270: West.
1185 if (dx > 0) {
1186     angle += 90;
1187 } else {
1188     angle += 270;
1189 }
1190 // Divide into 16 quads.
1191 var quad = Math.round(angle / 360 * 16);
1192 if (quad == 16) {
1193     quad = 15;
1194 }
1195 // Display correct Pegman sprite.
1196 pegSpin.style.backgroundColor = (-quad * Robotmaze.PEGMAN_WIDTH) + 'px 0px';
1197 };
1198
1199 /**
1200  * Schedule the animations for a move or turn.
1201  * @param {!Array.<number>} startPos X, Y and direction starting points.
1202  * @param {!Array.<number>} endPos X, Y and direction ending points.
1203  */
1204 Robotmaze.schedule = function(startPos, endPos) {
1205     var deltas = [(endPos[0] - startPos[0]) / 4,
1206                 (endPos[1] - startPos[1]) / 4,
1207                 (endPos[2] - startPos[2]) / 4];
1208     Robotmaze.displayPegman(startPos[0] + deltas[0],
1209                             startPos[1] + deltas[1],
1210                             Robotmaze.constrainDirection16(startPos[2] + deltas[2]));
1211     Robotmaze.pidList.push(setTimeout(function() {
1212         Robotmaze.displayPegman(startPos[0] + deltas[0] * 2,
1213                                 startPos[1] + deltas[1] * 2,
1214                                 Robotmaze.constrainDirection16(startPos[2] + deltas[2] * 2));
1215     }, Robotmaze.stepSpeed));
1216     Robotmaze.pidList.push(setTimeout(function() {
1217         Robotmaze.displayPegman(startPos[0] + deltas[0] * 3,
1218                                 startPos[1] + deltas[1] * 3,
1219                                 Robotmaze.constrainDirection16(startPos[2] + deltas[2] * 3));
1220     }, Robotmaze.stepSpeed * 2));
1221     Robotmaze.pidList.push(setTimeout(function() {
1222         Robotmaze.displayPegman(endPos[0], endPos[1],
1223                                 Robotmaze.constrainDirection16(endPos[2]));
1224     }, Robotmaze.stepSpeed * 3));
1225 };
1226
1227 /**
1228  * Schedule the animations and sounds for a failed move.
1229  * @param {boolean} forward True if forward, false if backward.
1230  */
1231 Robotmaze.scheduleFail = function(forward) {
1232     var deltaX = 0;
1233     var deltaY = 0;
1234     switch (Robotmaze.pegmanD) {
1235         case Robotmaze.DirectionType.NORTH:
1236             deltaY = -1;
1237             break;
1238         case Robotmaze.DirectionType.EAST:
1239             deltaX = 1;
1240             break;
1241         case Robotmaze.DirectionType.SOUTH:

```

```

1242     deltaY = 1;
1243     break;
1244     case Robotmaze.DirectionType.WEST:
1245         deltaX = -1;
1246         break;
1247 }
1248 if (!forward) {
1249     deltaX = -deltaX;
1250     deltaY = -deltaY;
1251 }
1252 if (Robotmaze.SKIN.crashType == Robotmaze.CRASH_STOP) {
1253     // Bounce bounce.
1254     deltaX /= 4;
1255     deltaY /= 4;
1256     var direction16 = Robotmaze.constrainDirection16(Robotmaze.pegmanD * 4);
1257     Robotmaze.displayPegman(Robotmaze.pegmanX + deltaX,
1258                             Robotmaze.pegmanY + deltaY,
1259                             direction16);
1260     BlocklyGames.workspace.getAudioManager().play('fail', 0.5);
1261     Robotmaze.pidList.push(setTimeout(function() {
1262         Robotmaze.displayPegman(Robotmaze.pegmanX,
1263                                 Robotmaze.pegmanY,
1264                                 direction16);
1265     }, Robotmaze.stepSpeed));
1266     Robotmaze.pidList.push(setTimeout(function() {
1267         Robotmaze.displayPegman(Robotmaze.pegmanX + deltaX,
1268                                 Robotmaze.pegmanY + deltaY,
1269                                 direction16);
1270         BlocklyGames.workspace.getAudioManager().play('fail', 0.5);
1271     }, Robotmaze.stepSpeed * 2));
1272     Robotmaze.pidList.push(setTimeout(function() {
1273         Robotmaze.displayPegman(Robotmaze.pegmanX, Robotmaze.pegmanY, direction16);
1274     }, Robotmaze.stepSpeed * 3));
1275 } else {
1276     // Add a small random delta away from the grid.
1277     var deltaZ = (Math.random() - 0.5) * 10;
1278     var deltaD = (Math.random() - 0.5) / 2;
1279     deltaX += (Math.random() - 0.5) / 4;
1280     deltaY += (Math.random() - 0.5) / 4;
1281     deltaX /= 8;
1282     deltaY /= 8;
1283     var acceleration = 0;
1284     if (Robotmaze.SKIN.crashType == Robotmaze.CRASH_FALL) {
1285         acceleration = 0.01;
1286     }
1287     Robotmaze.pidList.push(setTimeout(function() {
1288         BlocklyGames.workspace.getAudioManager().play('fail', 0.5);
1289     }, Robotmaze.stepSpeed * 2));
1290     var setPosition = function(n) {
1291         return function() {
1292             var direction16 = Robotmaze.constrainDirection16(Robotmaze.pegmanD * 4 +
1293                                                             deltaD * n);
1294             Robotmaze.displayPegman(Robotmaze.pegmanX + deltaX * n,
1295                                     Robotmaze.pegmanY + deltaY * n,
1296                                     direction16,
1297                                     deltaZ * n);
1298             deltaY += acceleration;
1299         };
1300     };
1301     // 100 frames should get Pegman offscreen.
1302     for (var i = 1; i < 100; i++) {
1303         Robotmaze.pidList.push(setTimeout(setPosition(i),
1304                                           Robotmaze.stepSpeed * i / 2));
1305     }
1306 }
1307 };
1308
1309 /**
1310  * Schedule the animations and sound for a victory dance.

```

```

1311     * @param {boolean} sound Play the victory sound.
1312     */
1313 Robotmaze.scheduleFinish = function(sound) {
1314     var direction16 = Robotmaze.constrainDirection16(Robotmaze.pegmanD * 4);
1315     Robotmaze.displayPegman(Robotmaze.pegmanX, Robotmaze.pegmanY, 16);
1316     if (sound) {
1317         BlocklyGames.workspace.getAudioManager().play('win', 0.5);
1318     }
1319     Robotmaze.stepSpeed = 150; // Slow down victory animation a bit.
1320     Robotmaze.pidList.push(setTimeout(function() {
1321         Robotmaze.displayPegman(Robotmaze.pegmanX, Robotmaze.pegmanY, 18);
1322     }, Robotmaze.stepSpeed));
1323     Robotmaze.pidList.push(setTimeout(function() {
1324         Robotmaze.displayPegman(Robotmaze.pegmanX, Robotmaze.pegmanY, 16);
1325     }, Robotmaze.stepSpeed * 2));
1326     Robotmaze.pidList.push(setTimeout(function() {
1327         Robotmaze.displayPegman(Robotmaze.pegmanX, Robotmaze.pegmanY, direction16);
1328     }, Robotmaze.stepSpeed * 3));
1329 };
1330
1331 /**
1332  * Display Pegman at the specified location, facing the specified direction.
1333  * @param {number} x Horizontal grid (or fraction thereof).
1334  * @param {number} y Vertical grid (or fraction thereof).
1335  * @param {number} d Direction (0 - 15) or dance (16 - 17).
1336  * @param {number=} opt_angle Optional angle (in degrees) to rotate Pegman.
1337  */
1338 Robotmaze.displayPegman = function(x, y, d, opt_angle) {
1339     var pegmanIcon = document.getElementById('pegman');
1340     pegmanIcon.setAttribute('x',
1341         x * Robotmaze.SQUARE_SIZE - d * Robotmaze.PEGMAN_WIDTH + 1);
1342     pegmanIcon.setAttribute('y',
1343         Robotmaze.SQUARE_SIZE * (y + 0.5) - Robotmaze.PEGMAN_HEIGHT / 2 - 8);
1344     if (opt_angle) {
1345         pegmanIcon.setAttribute('transform', 'rotate(' + opt_angle + ', ' +
1346             (x * Robotmaze.SQUARE_SIZE + Robotmaze.SQUARE_SIZE / 2) + ', ' +
1347             (y * Robotmaze.SQUARE_SIZE + Robotmaze.SQUARE_SIZE / 2) + ')');
1348     } else {
1349         pegmanIcon.setAttribute('transform', 'rotate(0, 0, 0)');
1350     }
1351
1352     var clipRect = document.getElementById('clipRect');
1353     clipRect.setAttribute('x', x * Robotmaze.SQUARE_SIZE + 1);
1354     clipRect.setAttribute('y', pegmanIcon.getAttribute('y'));
1355 };
1356
1357 /**
1358  * Display the look icon at Pegman's current location,
1359  * in the specified direction.
1360  * @param {!Robotmaze.DirectionType} d Direction (0 - 3).
1361  */
1362 Robotmaze.scheduleLook = function(d) {
1363     var x = Robotmaze.pegmanX;
1364     var y = Robotmaze.pegmanY;
1365     switch (d) {
1366         case Robotmaze.DirectionType.NORTH:
1367             x += 0.5;
1368             break;
1369         case Robotmaze.DirectionType.EAST:
1370             x += 1;
1371             y += 0.5;
1372             break;
1373         case Robotmaze.DirectionType.SOUTH:
1374             x += 0.5;
1375             y += 1;
1376             break;
1377         case Robotmaze.DirectionType.WEST:
1378             y += 0.5;
1379             break;

```

```

1380     }
1381     x *= Robotmaze.SQUARE_SIZE;
1382     y *= Robotmaze.SQUARE_SIZE;
1383     var deg = d * 90 - 45;
1384
1385     var lookIcon = document.getElementById('look');
1386     lookIcon.setAttribute('transform',
1387         'translate(' + x + ', ' + y + ') ' +
1388         'rotate(' + deg + ' 0 0) scale(.4)');
1389     var paths = lookIcon.getElementsByTagName('path');
1390     lookIcon.style.display = 'inline';
1391     for (var i = 0, path; (path = paths[i]); i++) {
1392         Robotmaze.scheduleLookStep(path, Robotmaze.stepSpeed * i);
1393     }
1394 };
1395
1396 /**
1397  * Schedule one of the 'look' icon's waves to appear, then disappear.
1398  * @param {!Element} path Element to make appear.
1399  * @param {number} delay Milliseconds to wait before making wave appear.
1400  */
1401 Robotmaze.scheduleLookStep = function(path, delay) {
1402     Robotmaze.pidList.push(setTimeout(function() {
1403         path.style.display = 'inline';
1404         setTimeout(function() {
1405             path.style.display = 'none';
1406         }, Robotmaze.stepSpeed * 2);
1407     }, delay));
1408 };
1409
1410 /**
1411  * Keep the direction within 0-3, wrapping at both ends.
1412  * @param {number} d Potentially out-of-bounds direction value.
1413  * @return {number} Legal direction value.
1414  */
1415 Robotmaze.constrainDirection4 = function(d) {
1416     d = Math.round(d) % 4;
1417     if (d < 0) {
1418         d += 4;
1419     }
1420     return d;
1421 };
1422
1423 /**
1424  * Keep the direction within 0-15, wrapping at both ends.
1425  * @param {number} d Potentially out-of-bounds direction value.
1426  * @return {number} Legal direction value.
1427  */
1428 Robotmaze.constrainDirection16 = function(d) {
1429     d = Math.round(d) % 16;
1430     if (d < 0) {
1431         d += 16;
1432     }
1433     return d;
1434 };
1435
1436 // Core functions.
1437
1438 /**
1439  * Attempt to move pegman forward or backward.
1440  * @param {number} direction Direction to move (0 = forward, 2 = backward).
1441  * @param {string} id ID of block that triggered this action.
1442  * @throws {true} If the end of the robotmaze is reached.
1443  * @throws {false} If Pegman collides with a wall.
1444  */
1445 Robotmaze.move = function(direction, id) {
1446     if (!Robotmaze.isPath(direction, null)) {
1447         Robotmaze.log.push(['fail_' + (direction ? 'backward' : 'forward'), id]);
1448         throw false;

```

```

1449     }
1450     // If moving backward, flip the effective direction.
1451     var effectiveDirection = Robotmaze.pegmanD + direction;
1452     var command;
1453     switch (Robotmaze.constrainDirection4(effectiveDirection)) {
1454     case Robotmaze.DirectionType.NORTH:
1455         Robotmaze.pegmanY--;
1456         command = 'north';
1457         break;
1458     case Robotmaze.DirectionType.EAST:
1459         Robotmaze.pegmanX++;
1460         command = 'east';
1461         break;
1462     case Robotmaze.DirectionType.SOUTH:
1463         Robotmaze.pegmanY++;
1464         command = 'south';
1465         break;
1466     case Robotmaze.DirectionType.WEST:
1467         Robotmaze.pegmanX--;
1468         command = 'west';
1469         break;
1470     }
1471     Robotmaze.log.push([command, id]);
1472 };
1473
1474 /**
1475  * Turn pegman left or right.
1476  * @param {number} direction Direction to turn (0 = left, 1 = right).
1477  * @param {string} id ID of block that triggered this action.
1478  */
1479 Robotmaze.turn = function(direction, id) {
1480     if (direction) {
1481         // Right turn (clockwise).
1482         Robotmaze.pegmanD++;
1483         Robotmaze.log.push(['right', id]);
1484     } else {
1485         // Left turn (counterclockwise).
1486         Robotmaze.pegmanD--;
1487         Robotmaze.log.push(['left', id]);
1488     }
1489     Robotmaze.pegmanD = Robotmaze.constrainDirection4(Robotmaze.pegmanD);
1490 };
1491
1492 /**
1493  * Is there a path next to pegman?
1494  * @param {number} direction Direction to look
1495  *     (0 = forward, 1 = right, 2 = backward, 3 = left).
1496  * @param {string} id ID of block that triggered this action.
1497  *     Null if called as a helper function in Robotmaze.move().
1498  * @return {boolean} True if there is a path.
1499  */
1500 Robotmaze.isPath = function(direction, id) {
1501     var effectiveDirection = Robotmaze.pegmanD + direction;
1502     var square;
1503     var command;
1504     switch (Robotmaze.constrainDirection4(effectiveDirection)) {
1505     case Robotmaze.DirectionType.NORTH:
1506         square = Robotmaze.map[Robotmaze.pegmanY - 1] &&
1507             Robotmaze.map[Robotmaze.pegmanY - 1][Robotmaze.pegmanX];
1508         command = 'look_north';
1509         break;
1510     case Robotmaze.DirectionType.EAST:
1511         square = Robotmaze.map[Robotmaze.pegmanY][Robotmaze.pegmanX + 1];
1512         command = 'look_east';
1513         break;
1514     case Robotmaze.DirectionType.SOUTH:
1515         square = Robotmaze.map[Robotmaze.pegmanY + 1] &&
1516             Robotmaze.map[Robotmaze.pegmanY + 1][Robotmaze.pegmanX];
1517         command = 'look_south';

```

```
1518         break;
1519     case Robotmaze.DirectionType.WEST:
1520         square = Robotmaze.map[Robotmaze.pegmanY][Robotmaze.pegmanX - 1];
1521         command = 'look_west';
1522         break;
1523     }
1524     if (id) {
1525         Robotmaze.log.push([command, id]);
1526     }
1527     return square !== Robotmaze.SquareType.WALL && square !== undefined;
1528 };
1529
1530 /**
1531  * Is the player at the finish marker?
1532  * @return {boolean} True if not done, false if done.
1533  */
1534 Robotmaze.notDone = function() {
1535     return Robotmaze.pegmanX !== Robotmaze.finish_.x || Robotmaze.pegmanY !==
1536         Robotmaze.finish_.y;
1537 };
1538 window.addEventListener('load', Robotmaze.init);
1539
```