```
1   //hoisted
2   function rand (min, max) {
3       // Pure JS!
4       return Math.floor(Math.random() * (max - min + 1)) + min;
5   }
6
7   var Puzzle = function(game, pic, square) {
8       // ==============
9       // Game variables
10      // ==============
11      this.game = game;
12      this.pic = pic;
13      this.square = square;
14      this.won = false;
15
16      //load source image to get image height/width properties
17      // Phaser v2.x.x replaced with Phaser III image
18      this.src_image = this.game.add.image(GAMEAPP.viewportWidth/2, GAMEAPP.
        viewportHeight/2, pic);
19      this.src_image.anchor.setTo(0.5);
20      // this.src_image.setOrigin(0.5)
21      this.src_image.visible = false;
22
23      var w = this.src_image.width;
24      var h = this.src_image.height;
25
26      //create a center anchor point
27      this.offsetX = (GAMEAPP.viewportWidth - w)/2;
28      this.offsetY = (GAMEAPP.viewportHeight - h)/2;
29
30      this.tile_width = Math.floor(w/this.square);
31      this.tile_height = Math.floor(h/this.square);
32
33      //data structure to handle jigsaw pieces
34      this.pieces = [];
35      this.slots = [];
36      this.background = {};
37      this.piece_list = {};
38      // End game variables
39      // ==============
40      //Setup Background Game Board
41      for (var i = 0; i < this.square;i++) {
42        for (var j = 0; j < this.square;j++) {
43                  // Phaser v2.x.x
44                  var slot = this.game.add.sprite(this.offsetX+j*this.
                    tile_width,this.offsetY+i*this.tile_height, this.makeBox(
                    this.tile_width, this.tile_height));
45                  // Game Object data
46                  slot.j = j;
47                  slot.i = i;
48                  this.background[j+'_'+i] = slot;
49                  this.slots.push(slot);
50              }
51          }
52      // ==============
53      // Offsets for puzzle sides
54      //         0 = flat
55      //         1 = [mountains, pegs, nooks, nobs]
56      //        -1 = [valleys, pits, crannies, divots]
```

```
57
58          // only worry about the concaves and protrusions
59          var choice = [-1, 1];
60          // ==============
61          //Now cycle through the image rows (Y coordinates)
62          // Game Mechanics (data)
63          for (var i = 0; i < this.square; i++) {
64              //Now cycle through the image columes (X coordinates)
65              for (var j = 0; j < this.square; j++) {
66
67                  //reset the jigsaw sides (clockwise) for this piece
68                  var sides = {ts: 0, rs: 0, bs: 0, ls: 0};
69
70                  //choose complement to fit this piece top side
71                  if (this.piece_list[j+'_'+(i-1)] !== undefined) {
72                      sides.ts = this.piece_list[j+'_'+(i-1)].bottom_side * -1;
73                  } else {
74                      sides.ts = choice[rand(0,1)];
75                  }
76
77                  //choose complement to fit this piece on the left
78                  if (this.piece_list[(j-1)+'_'+i] !== undefined) {
79                      sides.ls = this.piece_list[(j-1)+'_'+i].right_side * -1;
80                  } else {
81                      sides.ls = choice[rand(0,1)];
82                  }
83
84                  //bottom of this piece
85                  sides.bs = choice[rand(0,1)];
86
87                  //right of this piece
88                  sides.rs = choice[rand(0,1)];
89
90                  if (j === (this.square -1)) { sides.rs = 0; }
91                  if (i === 0) { sides.ts = 0; }
92                  if (i === (this.square - 1)) { sides.bs = 0; }
93                  if (j === 0) { sides.ls = 0; }
94                  // ==============
95                  // Game Mechanisms
96                  var piece = new PuzzlePiece(this.game, this.offsetX+j*this.
                    tile_width, this.offsetY+i*this.tile_height, j, i, this.
                    tile_width, this.tile_height,pic, sides);
97                  //NEW OLOO style
98                  //var piece = Object.create(PuzzlePiece(this.game,
                    this.offsetX+j*this.tile_width,
                    this.offsetY+i*this.tile_height, j, i, this.tile_width,
                    this.tile_height,pic, sides));
99                  piece.events.onDragStart.add(this.onDragStart, this);
100                 piece.events.onDragStop.add(this.onDragStop, this);
101
102                 this.pieces.push(piece);
103                 this.piece_list[j+'_'+i] = piece;
104
105             }
106         }
107     };
108
109     //force fitting OO onto JavaScript; use OLOO instead
110     Puzzle.prototype = Puzzle.prototype.constructor = Puzzle;
```

```
111
112   //force fitting OO onto JavaScript; use OLOO instead
113   // Game Mechanisms
114   Puzzle.prototype = {
115       onDragStart: function(sprite, pointer) {
116           // Phaser v2.x.x replaced with Phaser III display list
117           this.game.world.bringToTop(sprite);
118       },
119       // Game Mechanisms
120       onDragStop: function(piece, pointer) {
121
122           var slot = this.background[piece.j+'_'+piece.i];
123
124           if (Phaser.Rectangle.intersects(piece.getBounds(), slot.getBounds
          ())) {
125               //Disable and place piece
126               this.game.world.sendToBack(piece);
127               slot.visible = false;
128               piece.inputEnabled = false;
129               piece.input.enableDrag(false);
130               piece.x = piece.initialX;
131               piece.y = piece.initialY;
132               this.slots.forEach(function(slot) {
133                   this.game.world.sendToBack(slot);
134               },this);
135
136               this.won = this.checkWin();
137           }
138
139       },
140       // Game Mechanics
141       checkWin: function() {
142           var won = true;
143           for(var i=0; i< this.pieces.length;i++) {
144               if (this.pieces[i].x !== this.pieces[i].initialX && this.
              pieces[i].y !== this.pieces[i].initialY) {
145                   won = false;
146                   }
147           }
148           return won;
149       },
150       // Game Mechanisms
151       scatter: function() {
152           for (var s=0; s < this.pieces.length;s++) {
153               var piece = this.pieces[s];
154               piece.x = rand(0, GAMEAPP.viewportWidth-this.tile_width/2);
155               piece.y = rand(this.tile_height/2, GAMEAPP.viewportHeight-this
              .tile_height/2);
156
157           }
158       },
159       // Game Mechanisms
160       destroy: function() {
161           this.slots.forEach(function(slot) {
162               slot.destroy();
163           },this);
164           this.pieces.forEach(function(piece) {
165               piece.destroy();
166           },this);
```

```
167         },
168         // Game Mechanisms
169         preview_toggle: function() {
170             if (this.src_image.visible === false) {
171                 this.src_image.visible = true;
172                 this.game.world.bringToTop(this.src_image);
173                 this.pieces.forEach(function(piece) {
174                     piece.visible = false;
175                 },this);
176             }else {
177                 this.src_image.visible = false;
178                 this.pieces.forEach(function(piece) {
179                     piece.visible = true;
180                 },this);
181             }
182         },
183         // Game Mechanisms
184         makeBox: function(x,y) {
185           var bmd = this.game.add.bitmapData(x, y);
186           bmd.ctx.beginPath();
187           bmd.ctx.rect(0, 0, x, y);
188           bmd.ctx.fillStyle = '#202020';
189                 bmd.ctx.lineStyle = 4;
190           bmd.ctx.strokeStyle = '#ff00ff';
191           bmd.ctx.fill();
192           return bmd;
193         },
194     };
195
```