```
1   // (source code) https://phaser.io/examples/v2/games/sliding-puzzle
2   // article: http://zofiakorcz.pl/sliding-puzzle-game-phaser
3   // GitHub: https://github.com/Calanthe/PhaserSlidingPuzzle
4   // Live Demo v2.3.0: http://zofiakorcz.pl/PhaserSlidingPuzzle/
5
6   // var game = new Phaser.Game(800, 600, Phaser.CANVAS, 'phaser-example',
    { preload: preload, create: create });
7   // Add Phaser III config object
8   // var config = {};
9
10  // Move game variables to Game Mechanics Component
11  // remove hard-coded variables
12  var PIECE_WIDTH = 200,  // image must be evenly divisible by piece size.
13      PIECE_HEIGHT = 200, // only square permitted
14      BOARD_COLS,         // = imageUsed / config.width
15      BOARD_ROWS;         // = imageUsed / config.height
16  var piecesGroup,        // puzzle grouped as Dynamic
17      piecesAmount,       // total pieces for victory
18      shuffledIndexArray = [];// recorded image shuffle
19  // abstract other variables below as required
20
21  // Phaser Essential Functions
22  function preload() {
23      // Phaser v2.x.x into Phaser III
24      game.load.spritesheet("background",
        "assets/games/sliding-puzzle/bl.jpg", PIECE_WIDTH, PIECE_HEIGHT);
25      // into Phaser III
26      // this.load.spritesheet("background",
        "assets/games/sliding-puzzle/bl.jpg", PIECE_WIDTH, PIECE_HEIGHT);
27  }
28
29  function create() {
30      // consolidated creation; must deconstruct.
31      // mixture of game data and visual display
32      // split into machanics vs. mechanisms
33      prepareBoard();
34  }
35
36  // Supporting Game Functions
37  function prepareBoard() {
38      // split into machanics vs. mechanisms
39      // working variables for build process
40      var piecesIndex = 0,
41          i, j,
42          piece;
43      // ==========
44      // move to game mechanics component
45      BOARD_COLS = Math.floor(game.world.width / PIECE_WIDTH);
46      BOARD_ROWS = Math.floor(game.world.height / PIECE_HEIGHT);
47      // game data stored in Game Mechanics
48      piecesAmount = BOARD_COLS * BOARD_ROWS;
49      // game data stored in Game Mechanics
50      shuffledIndexArray = createShuffledIndexArray();
51      // ==========
52      // game mechanism unique to JS Gaming Framework
53      // Phaser v2.x.x
54      piecesGroup = game.add.group();
55      // Phaser III
56      // piecesGroup = this.physics.add.group();
```



```
57        // ======================
58        // Design Considerations:
59        // ======================
60        // into Phaser III Dynamic Group or "Containers" (an array of game
          objects) or Grid?!
61        // - Groups themselves aren't displayable, and can't be positioned,
          rotated, scaled, or hidden.
62        // -
          https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Container
          .html
63        // - Containers are used when you want two or more game objects to
          “behave as one flock”.
64        // - Container has position, angle, alpha, visible, but group does not.
65        // - Container controls properties of children, but group won't.
66        // -
          http://labs.phaser.io/edit.html?src=src\game%20objects\container\add%20
          array%20of%20sprites%20to%20container.js
67        // scene.physics.add.group(children, config);
68        // - A game object could be added to many group, but only added to a
          container (exclusive mode).
69        //
          https://rexrainbow.github.io/phaser3-rex-notes/docs/site/arcade-gameobj
          ect/#group
70        // https://rexrainbow.github.io/phaser3-rex-notes/docs/site/container/
71        //
          https://rexrainbow.github.io/phaser3-rex-notes/docs/site/containerlite/
          #children-group
72        // ======================
73        // A Phaser III Tilemap isn't a display object. It holds data about
          the map and can contain one or more layers, which are the display
          objects that actually render Tile objects. They come in two flavors:
          `StaticTilemapLayer` & `DynamicTilemapLayer`.
74        // - A `StaticTilemapLayer` is fast, but the tiles in that layer
          can’t be modified and can’t render per-tile effects like flipping or
          tint.
75        // * A `DynamicTilemapLayer` trades some speed for the flexibility
          and power of manipulating individual tiles. Unlike a Static Tilemap
          Layer, you can apply per-tile effects like tint or alpha, and you can
          change the tiles in a DynamicTilemapLayer.
76        // new DynamicTilemapLayer(scene, tilemap, layerIndex, tileset [, x]
          [, y])
77        //
78        //You can mix static and dynamic layers together in the same map.
79        // ======================
80        // Grid:
          https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Grid.html
81        // The Grid Shape is a Game Object that can be added to a Scene,
          Group or Container. You can treat it like any other Game Object in
          your game, such as tweening it, scaling it, or enabling it for input
          or physics. It provides a quick and easy way for you to render this
          shape in your game without using a texture, while still taking
          advantage of being fully batched in WebGL.
82        // This shape supports only fill colors and cannot be stroked.
83        // A Grid Shape allows you to display a grid in your game, where you
          can control the size of the grid as well as the width and height of
          the grid cells. You can set a fill color for each grid cell as well
          as an alternate fill color. When the alternate fill color is set then
          the grid cells will alternate the fill colors as they render,
          creating a chess-board effect. You can also optionally have an
```



```
        outline fill color. If set, this draws lines between the grid cells
        in the given color. If you specify an outline color with an alpha of
        zero, then it will draw the cells spaced out, but without the lines
        between them.
84      // ======================
85      // move to Game Mechanics Component
86      for (i = 0; i < BOARD_ROWS; i++){
87          for (j = 0; j < BOARD_COLS; j++){
88              if (shuffledIndexArray[piecesIndex]) {
89                  // Phaser v2.x.x  display
90                  piece = piecesGroup.create(j * PIECE_WIDTH, i *
                    PIECE_HEIGHT, "background",
                    shuffledIndexArray[piecesIndex]);
91                  // into Phaser III Dynamic Group
92
93              } else { //initial position of black piece
94                  // Phaser v2.x.x  display
95                  piece = piecesGroup.create(j * PIECE_WIDTH, i *
                    PIECE_HEIGHT);
96                  piece.black = true;
97              }
98              // about the Game Object (data):
99              piece.name = 'piece' + i.toString() + 'x' + j.toString();
100             // current grid location
101             piece.currentIndex = piecesIndex;
102             // final resting location
103             piece.destIndex = shuffledIndexArray[piecesIndex];
104             // Phaser v2.x.x
105             piece.inputEnabled = true;
106             // into Phaser III Dynamic Group
107             // piece.setInteractive();
108             // Game Object input responce from mouse/touch
109             // Phaser v2.x.x
110             piece.events.onInputDown.add(selectPiece, this);
111             // into Phaser III
112             // this.input.on("pointerdown", selectPiece, this);
113             piece.posX = j;
114             piece.posY = i;
115             piecesIndex++;
116         }
117     }
118
119 }
120 // Phaser v2.x.x display mechanism
121 function selectPiece(piece) {
122
123     var blackPiece = canMove(piece);
124
125     //if there is a black piece in neighborhood
126     if (blackPiece) {
127         // Phaser v2.x.x  display
128         movePiece(piece, blackPiece);
129     }
130
131 }
132 // Phaser v2.x.x display mechanism
133 function canMove(piece) {
134
135     var foundBlackElem = false;
```

```
136         // Phaser v2.x.x display mechanism
137         piecesGroup.children.forEach(function(element) {
138             if (element.posX === (piece.posX - 1) && element.posY ===
            piece.posY && element.black ||
139                 element.posX === (piece.posX + 1) && element.posY ===
                piece.posY && element.black ||
140                 element.posY === (piece.posY - 1) && element.posX ===
                piece.posX && element.black ||
141                 element.posY === (piece.posY + 1) && element.posX ===
                piece.posX && element.black) {
142                 foundBlackElem = element;
143                 return;
144             }
145         });
146 
147         return foundBlackElem;
148     }
149     // Phaser v2.x.x display mechanism
150     function movePiece(piece, blackPiece) {
151 
152         var tmpPiece = {
153             posX: piece.posX,
154             posY: piece.posY,
155             currentIndex: piece.currentIndex
156         };
157         // Phaser v2.x.x display mechanism - tween
158         game.add.tween(piece).to({x: blackPiece.posX * PIECE_WIDTH, y:
            blackPiece.posY * PIECE_HEIGHT}, 300, Phaser.Easing.Linear.None, true);
159 
160         //change places of piece and blackPiece
161         piece.posX = blackPiece.posX;
162         piece.posY = blackPiece.posY;
163         piece.currentIndex = blackPiece.currentIndex;
164         piece.name ='piece' + piece.posX.toString() + 'x' +
            piece.posY.toString();
165 
166         //piece is the new black
167         blackPiece.posX = tmpPiece.posX;
168         blackPiece.posY = tmpPiece.posY;
169         blackPiece.currentIndex = tmpPiece.currentIndex;
170         blackPiece.name ='piece' + blackPiece.posX.toString() + 'x' +
            blackPiece.posY.toString();
171 
172         //after every move check if puzzle is completed
173 
174         checkIfFinished();  // move to Game Mechanics Component
175     }
176     // move to Game Mechanics Component
177     function checkIfFinished() {
178 
179         var isFinished = true;
180         // validates data from Game Mechanism
181         piecesGroup.children.forEach(function(element) {
182             if (element.currentIndex !== element.destIndex) {
183                 isFinished = false;
184                 return;
185             }
186         });
187 
```

```
188         if (isFinished) {
189             // tell Game Mechanism
190             showFinishedText();
191         }
192 
193     }
194     // Game Mechanism - display WON! or move to new "GameOver" Phase
195     function showFinishedText() {
196         // already in main.js
197         var style = { font: "40px Arial", fill: "#000", align: "center"};
198         // Phaser v2.x.x  display
199         var text = game.add.text(game.world.centerX, game.world.centerY,
            "Congratulations! \nYou made it!", style);
200         // into Phaser III
201         // var text = this.add.text(config.width/2, config.height/2,
            "Congratulations! \nYou made it!", style);
202         // Phaser v2.x.x  display
203         text.anchor.set(0.5);
204         // into Phaser III
205         // text.setOrigin(0.5);
206 
207     }
208     // move to Game Mechanics (data)
209     function createShuffledIndexArray() {
210 
211         var indexArray = [];
212 
213         for (var i = 0; i < piecesAmount; i++) {
214             indexArray.push(i);
215         }
216 
217         return shuffle(indexArray);
218 
219     }
220     // move to Game Mechanics (data)
221     function shuffle(array) {
222 
223         var counter = array.length,
224             temp,
225             index;
226 
227         while (counter > 0){
228             index = Math.floor(Math.random() * counter);
229 
230             counter--;
231             // standard Fisher Yates shuffle
232             // https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle
233             temp = array[counter];
234             array[counter] = array[index];
235             array[index] = temp;
236         }
237 
238         return array;
239 
240     }
241 
```