JavaScript Project 1.1

Prepared by Nasreddine Hallam

Abstract

The purpose is to create and manage instances of the 8-tile puzzle game. Each game instance has a game setting (player name, level) and a game live progress (duration and number of moves). Each game instance (whether solved or abandoned) is saved in a table score showing player name, game outcome, number of moves and duration.

Keywords: html, css, JavaScript, DOM

Table of Contents

1. [Project files system and hierarchy 2](#_bookmark0)
2. [Part 1 HTML, CSS, Javascript for TABS. 2](#_bookmark1)
3. [Part2. The use case diagram 3](#_bookmark2)
   1. [Th user name and the dimension 3](#_bookmark3)
   2. [The cancel button 3](#_bookmark4)
   3. [If the user plays and wins 4](#_bookmark5)
   4. [If the user clicks on the second TAB. 4](#_bookmark6)
4. [The sequence diagram 4](#_bookmark7)
   1. [The function init 5](#_bookmark8)
   2. [How/when to call the init function 5](#_bookmark9)
5. [Main objects 5](#_bookmark10)
   1. [The Utility 5](#_bookmark11)
   2. [the PlayerManager 6](#_bookmark12)
   3. [Tile 6](#_bookmark13)
   4. [PuzzleGame 6](#_bookmark14)
   5. [mainProgram 7](#_bookmark15)
6. [Some GUI 8](#_bookmark16)

[Appendices 10](#_bookmark17)

[Appendix 1 (style.css specs) 10](#_bookmark18)

[Appendix 2 (tabs.css specs) 11](#_bookmark19)

### You will need to do your outmost best to follow the specifications.

### Some requirements are strict, some are flexible (see optional).

# Project files system and hierarchy

|  |  |
| --- | --- |
| *Figure 1Project file system and hierarchy* | *Figure 2. main HTML UI* |

* Create the folder Puzzle in your sonic JS423.
* Inside you create the directories js, sounds, and style as shown in figure above.

# Part 1 HTML, CSS, Javascript for TABS.

The main UI interface code must have the structure as shown in Figure2.

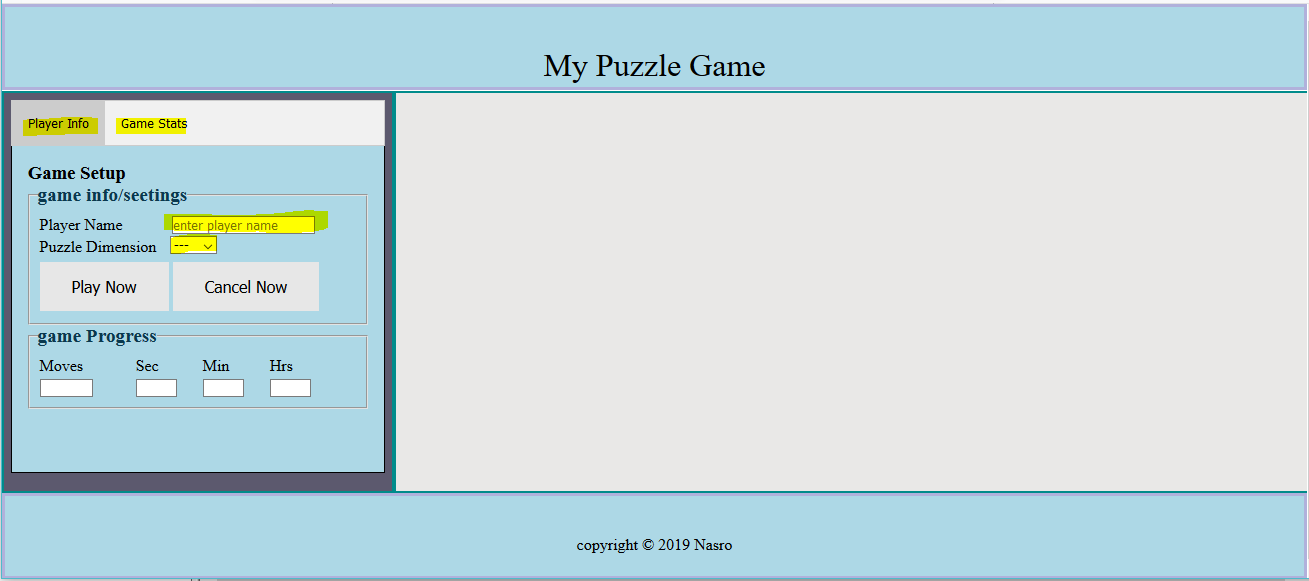
Write the code for the game UI (html code) in the root folder (Puzzle). You name the html file **my\_puzzle.html**.

The css code styling in the following file

* The **style.css** contains the main html styling. Refer to appendix 1.
* The **tabs.css** contains the styling for the two tabs in the section leftSection, as well the styling of the score table. Refer to [appendix 2.](#_bookmark19)

The JS code for manipulating the two tabs.

* Write the JS code in a file you call **tabs.js**.
* The two tabs are buttons and under them are two divs with display:none (not shown).
* When one tab button is clicked, one the display of div is set to block, while the rest are set to none.



*Figure 3 - Main GUI*

# Part2. The use case diagram

The game and score managements are handled in this file (puzzle.js).

The **use-case diagram,** shown on the right in Figure 3 and explained below, depicts the user interaction with the system (the game).

## The user name and the dimension

The user enters the name and the board dimension. The **dimension is set to 3** (later extend it to 4 and 5 dimension).

Once the two data entered, the play button is enabled and can be clicked. If the user remove any of the entered data, the button must be disabled again.

When the play button is enabled and is clicked: *Figure 4- The Project Use Case Diagram*

* + - The cancel button is enabled.
    - The puzzleBoard is drawn
    - The chrono starts (counting).

## If the user plays and hit the cancel button

The user can click on the cancel button anytime. Once clicked, the user name, board dimension, number of moves and the duration in seconds (chrono) are used as internal information to instantiate a Player. One of the Player property attribute (called **theStatus)** must be set with the value “**cancelled**”.   
The player is then pushed in a global array called **listPlayerArr**.

## 3.3 If the user plays and wins

The **Utility** object plays the firework.mp3 sound. Then, the name, board dimension, number of moves and the duration in seconds (chrono) are used as internal information to instantiate a Player. The Player property attribute **theStatus** is set with the value “**success**”.   
The player is then pushed in a global array called **listPlayerArr**.

## If the user clicks on the second TAB.

he second tab must be functional. Show all scores thus far.

# The sequence diagram

UML sequence diagram is an interaction diagram depicting how operations are carried out.   
Specifically,

**User**

**Tile**

**PuzzleGam**

**Utility**

**1. Enter name,**

**dimension**

1. **Choose Play**
2. **request random numbers**

**Or request sampleEasyBoardTests**

1. **Create PuzzleBoard**

**of Tiles 6. Process move Redraw PuzzleBoard**

1. **Click to move 6.a Request beep**

**sound (wrong move)**

**6.b Request firework sound (goal state)**

**6. terminateGame(win) 6. Push player with**

**Update**

**Stats**

**(winner state)**

**info to playerLists**

**7. Cancel Game**

**8. terminateGame(win)**

**Update Stats (cancel state)**

**8. Push player with**

**info to playerLists**

**9. Show scores 9. Loop through**

**playerLists**

**PlayerManager**

* it captures interactions between objects
* shows the order of the interaction using a vertical axis

The main objects are Utility, Player, PlayerManagement, Tile, and PuzzleGame.

However, before specifying in detail these objects, the function init and when it is executed are explained below.

## The function init

Define init as follows:

* + - instantiate objects out of **Utility** object and **PlayerManager** object.
    - attach an event listeners to
      * the button play -> handler **mainProgram**.
      * the button cancel-> handler **cancelPuzzlePlay**.
    - attach an event listeners to player name and game level.
    - and event listeners to anything you deem necessary

## How/when to call the init function

Write the code/instruction that calls init function when the content of the DOM is loaded. It is not a must but I would like to see init definition and its call at the end of the file.

# Main objects

## The Utility

Defines the following functions that are used by other objects (mainly the object function PuzzleGame)

* + 1. **generateRandomNumber** (minValue, maxValue): generate a random number between minValue and maxValue. Will be used by PuzzleGame to create the board initial random state.
    2. **playAudio**: plays the audio file for winning or whenever a wrong move is done.
    3. **sampleEasyBoardTests** : in case we start a game with a very easy state (2 or 3 moves away from the goal state). Used in testing.

If \_PUZZLE\_WIDTH is 3, then the array to return is :

puzzleBoardTest = [ [1, 2, 3], [4, 0, 6], [7, 5, 8]]. Remember the empty tile is represented by (0).

* + 1. **terminateGame** (theStatus): receives theStatus (success or cancelled). Stops the chrono using clearInterval and calls setGameInfo\_Stats.

It then empties the player name and the game level and resets nber of Moves, duration , seconds,minutes,hours to the value 0.

* + 1. **cancelPuzzlePlay** calls terminateGame with theStatus = ":cancelled" and disable button cancelBtnId'.
    2. **checkFormFilled**: if both playerName and puzzleLevelId are set then enable Play button (and make it green). Otherwise (at least one is not set) then disable Play button (and make it gray)..
    3. **enableButton** (btnId, theStatus, btnClass) : receives three params: the button id, the disable status (true or false, and the button class (green, orange, …). Called by checkFormFilled. It changes the disabled status of the button and uses setattribute function to set its class.
    4. **showChrono**: increment duration, use it to compute hours, minutes, seconds and set the html inputs accordingly.
    5. **showStats**: (do not do this now)

create a dynamic table (with class "tableSats") and loops through array listPlayers to show each player stat.

## the PlayerManager :

has an array listPlayers , and initializes gameCounter , gameDuration and nberMoves;.   
it also has **storeGameStats**(theStatus) method that

* + 1. get the values from inputs
    2. Instantiate an object player.
    3. Push the instantiated player into listPlayers
    4. Then update gameCounter, nberMoves , gameDuration.

## Tile

This is at the heart of the game. It is used inside the PuzzleGame defined below.

The constructor sets the following data members row, col, tileType, indexNumber.   
The tileType could be emptyTile (for 0) or filledType (for the rest), and indexNumber (0 to (9-1)) 9: is puzzleLevel2-1.

Add any function you deem necessary

## PuzzleGame

This is the core of the project. The data members of your constructor object are:

* this.puzzleWidth; this. puzzleBoard = [] ; this.goalState = [];

some function members are as follows:

* + 1. **createGoalState** ().

### createBoardStructure

Uses function generateRandomNumber OR sampleEasyBoardTests (from Utility) to create the board. Each element of puzzleBoard is a Tile object.

### drawPuzzleBoard

append or concatenate each Tile of puzzleBoard to the div (<div class="puzzleBoard">').

The <div class="puzzleBoard"> is then added as innerHTML or a child to element ‘checkBoardId' Optional: you can resize dynamically your tiles (based on the game level).

* + 1. **swap2Tiles** (indexTile1, indexTile2)

swap the two elements (tiles) in puzzleBoard.

* + 1. **match2States** (state1, state2)

state1 and state2 are 2-dim array. Return true if the two states are identical

This function is used to see whether we reached the goal state.

* + 1. **getNeighboursIndicesArr** (arrayIndex)

return an array of four integers (indexNumbers) that are neighbor of the tile identified by arrayIndex.

/\* if a neighbour is off the grid, you may set it to -1. \*/

* + 1. **processClickTile** (arrayIndex)

get the neighbours of the tile in question (with param arrayIndex).

If one of the neighbours is the empty tile (its indexNumber is 0) then

* + - 1. swap2Tiles, nberMoves should be increased and redraw the board (drawPuzzleBoard). compute the heuristic (or match2States) and terminate the game if success.
      2. Otherwise wrong move (beep).

### computeNumberMisplaced

this could be used as heuristic function that counts how many tiles are not in their correct place. If the number is zero then it is a win. This is an alternative to function **match2States**.

## mainProgram

remember, this is the event handler attached to the click event on the play button. Once this is fired,

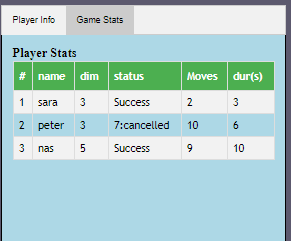
* the cancel button is enabled,
* create an instance object from PuzzlGame object with puzzleLevel passed as a parameter.
* Create the board structure by calling createBoardStructure();
* Start the chrono by calling showChrono from Utility object every second.

# Some GUI

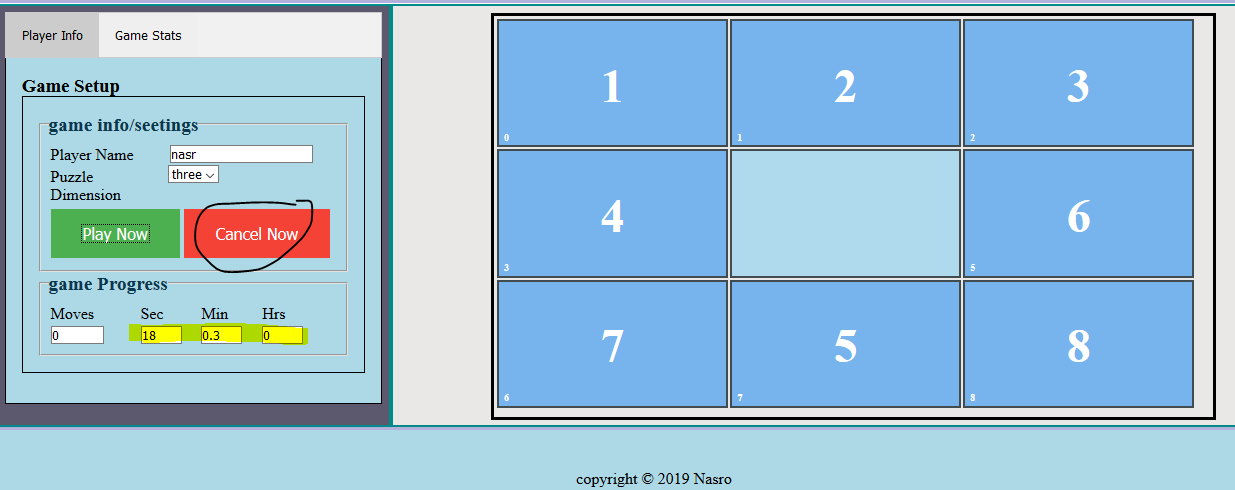
Initially,

|  |  |
| --- | --- |
| After you enter game setting (name, puzzle dimension), Play Now becomes enabled |  |

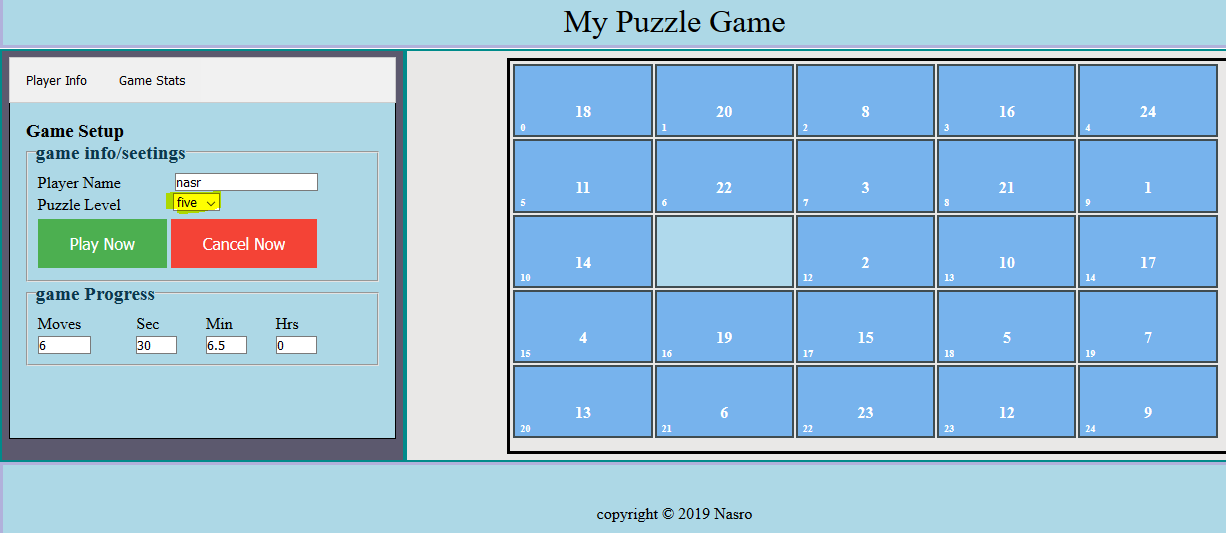
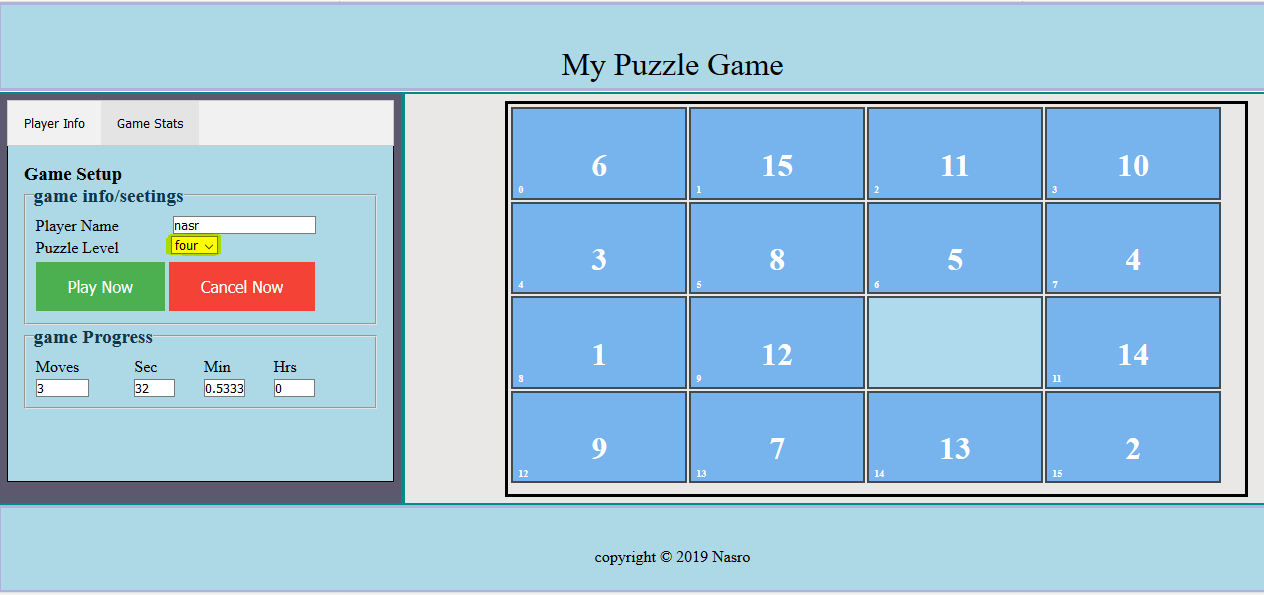
The Game Stats tab maintain a dynamic html table that append rows on the fly after each won /cancelled game.



Once you start play (Cancel button is enabled, time counter starts)



(optional) If you choose to allow 15 and 24 puzzle boards, you may want to dynamically adjust the width of your tales.



# Appendices

## Appendix 1 (style.css specs)

|  |  |
| --- | --- |
| For all box-sizing: border-box,  padding = margin = 0  common for header, footer  height:15%;  use proper styling for spacing, height,  common for leftSection and middleSection  . . .  border: 2px solid darkcyan;  use proper styling for spacing, height, floating or positioning,. . .  legend font-size: 1.2em;  color: #09384e;  font-weight: bold;  label width: 40%;  margin: 2px;  greenButton, blueButton, redButton , disabledButton{  background-color: #4CAF50; /\* Green \*/ #008CBA; /\* blue \*/ #f44336;} /\* Red \*/ #e7e7e7; /\* Gray \*/  . . .  use proper styling for spacing borders, cursor, …; | #checkBoardId float: left;  width: 100%;height: 100%;  .puzzleBoard {  /\* to be assigned to the inner tml of div#checkBoardId\*/  margin: 1px;  border: 3px solid black;  padding: 2px;  use proper styling for positioning, spacing borders, …  }  .filledTile, .emptyTile {  width: ???;  height: ???  text-align: center;  display: table-cell;  vertical-align: middle;  background-color: #77B3ED;  border: 2px solid #404c50;  color:white;  font-weight: bold;  use proper styling for spacing borders,  }  .emptyTile {  background-color: #afd9ec;  color: #afd9ec;  } |

## Appendix 2 (tabs.css specs)

/\*--------------------------\*/

/\*\*\*\*\* tabs structure \*\*\*\*/

/\*------ Stats table -------\*/

table.tableSats {

font-family: "Trebuchet MS", Arial, Helvetica, sans-serif;

border-collapse: collapse;

width: 100%;

}

use proper styling for td, th

use proper styling for table.tableSats tr:nth-child(XYZ), hover

/\*--------------------------\*