Michael Rooplall
CS389D Mobile Web Development

Sudoku in React Native

Final Project Code PDF

There are seven separate files required for the program to work.

`App.js`	Handles the application itself
`screens/MenuScreen.jsx`	Handles the menu
`screens/LevelsScreen.jsx`	Handles the list of level display for each difficulty
`screens/GameScreen.jsx`	Handles the actual sudoku gameplay
`game_levels.json`	Contains data for the game levels
`indexTracker.js`	Handles interoperability between the generator, board, and UI.
`sudokuGenerator.js`	Responsible for generating game levels and checking sudoku solutions

Alternatively, you can (**and should**) clone the GitHub repository here:

- https://github.com/DeveloperBlue/SudokuReactNative

Contents

App.js

```
import 'react-native-gesture-handler';
import { StatusBar } from 'expo-status-bar';
import React, {useState} from 'react';
import { StyleSheet, Text, TouchableHighlight, View, Image, FlatList } from
'react-native';

import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';

import MenuScreen from './screens/MenuScreen';
import LevelsScreen from './screens/LevelsScreen';
import GameScreen from './screens/GameScreen';

const Stack = createStackNavigator();


/* PAGINATION */

const App = () => {
      return (
            <NavigationContainer>
            <Stack.Navigator

                  initialRouteName="Menu"

                  screenOptions = {{
                        headerShown: false
                  }}>

                  <Stack.Screen
                  name="Menu"
                  component={MenuScreen}
                  options={{ title: 'Menu' }}
                  />
                  <Stack.Screen
                  name="Levels"
                  component={LevelsScreen}
                  options={{ title: 'Levels' }}
                  />
                  <Stack.Screen
                  name="Game"
                  component={GameScreen}
                  options={{ title: 'Game' }}
                  />

            </Stack.Navigator>
            </NavigationContainer>
      );
};

export default App;
```

screens/MenuScreen.jsx

```
import 'react-native-gesture-handler';
import { StatusBar } from 'expo-status-bar';
import React, {useState} from 'react';
import { StyleSheet, Text, TouchableHighlight, View, Image, FlatList } from
'react-native';

//

let game_levels = require("./../game_levels");

let difficulties = [];
let time_tracker = {};

for (let difficulty of Object.keys(game_levels)){
      time_tracker[difficulty] = {};
      difficulties.push({
            "key" : difficulty[0].toUpperCase() + difficulty.substring(1,
difficulty.length),
            "level_count" : Object.values(game_levels[difficulty]).length
      })
}

//

const MenuScreen = ({ navigation }) => {
      return (

            <View style={styles.menuContainer}>

                  <StatusBar style="auto" />

                  <View style={styles.logoView}>
                        <View style={styles.logoTop}>
                              <Text style={styles.logoTopText}>
                                    S
                              </Text>
                        </View>
                        <Text style={styles.logoBottom}>
                              SUDOKU
                        </Text>
                  </View>

                  <View style={styles.menuListView}>

                        <FlatList
                              data={difficulties}
                              renderItem={ ({item}) => {
                                    return (
                                          <TouchableHighlight
                                                activeOpacity={1}
                                                underlayColor={'#1565c0'}
                                                style={styles.menuButton}
                                                onPress={() => {

navigation.navigate('Levels', {
```

```
                                                difficulty :
item.key,
                                                level_count :
item.level_count,
                                                time_tracker :
time_tracker,
                                                game_levels :
game_levels
                                          })
                                    }}
                              >
                                    <View
style={styles.menuButtonView}>
                                          <Text
style={styles.menuButtonText}>
                                                {item.key}
                                          </Text>
                                          <Text


style={styles.menuButtonSubtext}>

{`0/${item.level_count}`}
                                          </Text>
                                    </View>


                              </TouchableHighlight>
                                    )
                              }}
                        >

                        </FlatList>

                  </View>
            </View>

      )
}

const styles = StyleSheet.create({
      container: {
            flex: 1,
            backgroundColor: '#fff',
      },

      // MENU PAGE
      menuContainer : {
            flex: 1,
            backgroundColor: '#fff',
            padding : 40,
      },

            // MENU LOGO

      logoView : {
            flex : 2,
            alignItems : "center",
            justifyContent : "center"
```

```
},

logoTop : {
      alignItems : "center",
      justifyContent : "center",
      textAlign : "center",
      backgroundColor : "#007acc",
      borderRadius : 20,
      width : 120,
      height : 120
},

logoTopText : {
      textAlign : "center",
      fontSize : 74,
      color : "#fff",
      fontWeight : "bold"
},

logoBottom : {
      fontSize : 32,
      fontWeight : "100"
},



      // MENU LIST & BUTTONS

menuListView : {
      flex : 3
},

menuButton : {
      alignContent : "center",
      justifyContent : "center",
      backgroundColor : "#007acc",
      height : 60,
      paddingLeft : 24,
      borderRadius : 2,
      marginBottom : 10
},

menuButtonText : {
      color : "#fff",
      fontSize : 22
},

menuButtonSubtext : {
      color : "#fff",
      fontSize : 12,
      position : "absolute",
      textAlign : "right",
      right : 6,
      bottom : -10
},

// LEVELS PAGE
```

```
levelsContainer : {
      flex: 1,
      backgroundColor: '#fff',
      padding : 20,
},

      // LEVELS HEADER
levelsHeader : {
      flex : 1,
      alignItems : "center",
      justifyContent : "center",
},

levelsHeaderTop : {
      fontSize : 42
},

levelsHeaderBottom : {
      fontSize : 24
},

      // LEVELS LIST & BUTTONS

levelsListView : {
      flex : 3
},

levelButton : {
      alignContent : "center",
      justifyContent : "center",
      backgroundColor : "#007acc",
      height : 60,
      paddingLeft : 20,
      borderRadius : 2,
      marginBottom : 5
},

levelButtonText : {
      color : "#fff",
      fontSize : 18
},

levelButtonSubtext : {
      color : "#fff",
      fontSize : 12,
      position : "absolute",
      textAlign : "right",
      right : 6,
      bottom : -12
},

      // LEVELS BACK BUTTON

levelsBackButton  :{
      marginTop : 40,
      width : 62,
```

```
            height : 62,
            borderRadius : 50,
            backgroundColor : "#007acc",
            alignItems : "center",
            justifyContent : "center",
            padding : 10,
      },

      levelsBackButtonIcon : {
            width : 32,
            height : 32
      }
})


export default MenuScreen;
```

screens/LevelsScreen.jsx

```
import 'react-native-gesture-handler';
import { StatusBar } from 'expo-status-bar';
import React, {useState} from 'react';
import { StyleSheet, Text, TouchableHighlight, View, Image, FlatList } from
'react-native';


const LevelsScreen = ({ navigation, route }) => {

      //

      let levels_list = [];

      let difficulty = route.params.difficulty;
      let level_count = route.params.level_count;
      let time_tracker = route.params.time_tracker;
      let game_levels = route.params.game_levels;

      for (let i = 1; i <= level_count; i++){
            levels_list.push({
                  key : i
            })
      }

      //

      return (

            <View style={styles.levelsContainer}>

                  <StatusBar style="auto" />

                  <TouchableHighlight
                        activeOpacity={1}
                        underlayColor={'#1565c0'}
                        style={styles.levelsBackButton}
                        onPress={() => {
                              navigation.navigate('Menu')
                        }}
                  >

                        <View>
                              <Image
                                    style = {styles.levelsBackButtonIcon}

source={require("./../assets/home.png")}
                              >

                              </Image>
                        </View>

                  </TouchableHighlight>

                  <View style={styles.levelsHeader}>
                        <Text
```

```
style={styles.levelsHeaderTop}>{difficulty}</Text>
                        <Text
style={styles.levelsHeaderBottom}>{`0/${level_count}`}</Text>
                  </View>


                  <View style={styles.levelsListView}>

                        <FlatList
                              data={levels_list}
                              renderItem={ ({item}) => {
                                    return (
                                          <TouchableHighlight
                                                activeOpacity={1}
                                                underlayColor={'#1565c0'}
                                                style={styles.levelButton}
                                                onPress={() => {

time_tracker[difficulty.toLowerCase()][item.key] = "0:00.00";

navigation.navigate('Game', {
                                                      difficulty :
difficulty,
                                                      level_no :
item.key,
                                                      game_levels :
game_levels
                                                })
                                          }}
                                    >
                                          <View
style={styles.levelButtonView}>
                                                <Text
style={styles.levelButtonText}>
                                                      {`Level
${item.key}`}
                                                </Text>
                                                <Text
style={styles.levelButtonSubtext}>
                                                      {
                                                            `${(typeof
time_tracker[difficulty.toLowerCase()][item.key] == "undefined") ? "" :
time_tracker[difficulty.toLowerCase()][item.key]}`
                                                      }
                                                </Text>
                                          </View>


                                          </TouchableHighlight>
                                    )
                              }}
                        >

                        </FlatList>

                  </View>
            </View>
```

```
      )
}

const styles = StyleSheet.create({
      container: {
            flex: 1,
            backgroundColor: '#fff',
      },

      // MENU PAGE
      menuContainer : {
            flex: 1,
            backgroundColor: '#fff',
            padding : 40,
      },

            // MENU LOGO

      logoView : {
            flex : 2,
            alignItems : "center",
            justifyContent : "center"
      },

      logoTop : {
            alignItems : "center",
            justifyContent : "center",
            textAlign : "center",
            backgroundColor : "#007acc",
            borderRadius : 20,
            width : 120,
            height : 120
      },

      logoTopText : {
            textAlign : "center",
            fontSize : 74,
            color : "#fff",
            fontWeight : "bold"
      },

      logoBottom : {
            fontSize : 32,
            fontWeight : "100"
      },



            // MENU LIST & BUTTONS

      menuListView : {
            flex : 3
      },

      menuButton : {
            alignContent : "center",
```

```
            justifyContent : "center",
            backgroundColor : "#007acc",
            height : 80,
            paddingLeft : 24,
            borderRadius : 2,
            marginBottom : 10
      },

      menuButtonText : {
            color : "#fff",
            fontSize : 22
      },

      menuButtonSubtext : {
            color : "#fff",
            fontSize : 12,
            position : "absolute",
            textAlign : "right",
            right : 10,
            bottom : -18
      },

      // LEVELS PAGE

      levelsContainer : {
            flex: 1,
            backgroundColor: '#fff',
            padding : 20,
      },

            // LEVELS HEADER
      levelsHeader : {
            flex : 1,
            alignItems : "center",
            justifyContent : "center",
      },

      levelsHeaderTop : {
            fontSize : 42
      },

      levelsHeaderBottom : {
            fontSize : 22,
            marginBottom : 10
      },

            // LEVELS LIST & BUTTONS

      levelsListView : {
            flex : 5
      },

      levelButton : {
            alignContent : "center",
            justifyContent : "center",
            backgroundColor : "#007acc",
            height : 60,
```

```
            paddingLeft : 20,
            borderRadius : 2,
            marginBottom : 5
      },

      levelButtonText : {
            color : "#fff",
            fontSize : 18
      },

      levelButtonSubtext : {
            color : "#fff",
            fontSize : 12,
            position : "absolute",
            textAlign : "right",
            right : 6,
            bottom : -12
      },

            // LEVELS BACK BUTTON

      levelsBackButton  :{
            marginTop : 40,
            width : 62,
            height : 62,
            borderRadius : 50,
            backgroundColor : "#007acc",
            alignItems : "center",
            justifyContent : "center",
            padding : 10,
      },

      levelsBackButtonIcon : {
            width : 32,
            height : 32
      }
})


export default LevelsScreen;
```

screens/GameScreen.jsx

```
import 'react-native-gesture-handler';
import { StatusBar } from 'expo-status-bar';
import React, {useState, useEffect} from 'react';
import { StyleSheet, Text, TouchableHighlight, Pressable, View, Image,
FlatList } from 'react-native';

import {sudoku_generator} from './../sudokuGenerator';
import indexTracker from "./../indexTracker";

//

const GameScreen = ({ navigation, route }) => {

      //
      let difficulty = route.params.difficulty;
      let level_no = route.params.level_no;
      let game_levels = route.params.game_levels;

      //


      const sudoku_string_master =
game_levels[difficulty.toLowerCase()][level_no];
      const [sudoku_string_play, setSudokuString] =
useState(sudoku_string_master + "");
      const sudoku_solution = sudoku_generator.solve(sudoku_string_play);

      const [history, updateHistory] = useState([]);

      //

      const [selectedGridItem, setSelectedGridItem] = useState(undefined);

      // Valid states: 'enabled', 'disabled', 'active'
      const [buttonHiglights, updateButtonHighlights] = useState({
            "b1" : "disabled",
            "b2" : "disabled",
            "b3" : "disabled",
            "b4" : "disabled",
            "b5" : "disabled",
            "b6" : "disabled",
            "b7" : "disabled",
            "b8" : "disabled",
            "b9" : "disabled",
      })

      const updateNumberButtons = () => {

            // if the 'selectedGridItem' is immutable, disable buttons
            // otherwise, highlight the number that 'selectedGridItem'
holds as a value

            if (selectedGridItem == undefined) return;

            let stringIndex = selectedGridItem.stringIndex;
```

```
            if (sudoku_string_master[stringIndex] !== "."){

                  // Item is immutable, disable the buttons.

                  updateButtonHighlights({
                        "b1" : "disabled",
                        "b2" : "disabled",
                        "b3" : "disabled",
                        "b4" : "disabled",
                        "b5" : "disabled",
                        "b6" : "disabled",
                        "b7" : "disabled",
                        "b8" : "disabled",
                        "b9" : "disabled",
                  })

            } else if (sudoku_string_play[stringIndex] == "."){
                  // Item is mutable, but has no value. Make all buttons
normal

                  updateButtonHighlights({
                        "b1" : "enabled",
                        "b2" : "enabled",
                        "b3" : "enabled",
                        "b4" : "enabled",
                        "b5" : "enabled",
                        "b6" : "enabled",
                        "b7" : "enabled",
                        "b8" : "enabled",
                        "b9" : "enabled",
                  })

            } else {
                  // Item is mutable, but has a current value. Highlight
that specific button.

                  let currentValue = sudoku_string_play[stringIndex];

                  (currentValue == 1) ? "active" : "enabled",

                  updateButtonHighlights({
                        "b1" : (currentValue == 1) ? "active" : "enabled",
                        "b2" : (currentValue == 2) ? "active" : "enabled",
                        "b3" : (currentValue == 3) ? "active" : "enabled",
                        "b4" : (currentValue == 4) ? "active" : "enabled",
                        "b5" : (currentValue == 5) ? "active" : "enabled",
                        "b6" : (currentValue == 6) ? "active" : "enabled",
                        "b7" : (currentValue == 7) ? "active" : "enabled",
                        "b8" : (currentValue == 8) ? "active" : "enabled",
                        "b9" : (currentValue == 9) ? "active" : "enabled",
                  })
            }

      }
```

```
      const handleUndoPressed = () => {

            if (history.length == 0) return;

            let undoItem = history.pop();
            updateHistory(history);

            // change item at undoItem.index from undoItem.newValue to
undoItem.previousValue;

            updateSudokuString(undoItem.stringIndex,
undoItem.previousValue, true);
      }

      const handleErasePressed = () => {
            // is selected item writeable and non-empty?
            // set value to empty, push to history

            if (selectedGridItem == undefined) return;
            let stringIndex = selectedGridItem.stringIndex;

            if (sudoku_string_master[stringIndex] !== ".") return; // Item
is immutable
            if (sudoku_string_play[stringIndex] == ".") return; // Item is
already clear

            updateSudokuString(stringIndex, ".");
      }

      const handleHintPressed = () => {

            // Check for empty items, add the correct value
                  // todo, animate a blink at that cell
            // If no empty items, compare current board to solution to
identify where player went wrong (or highlight overlaps red?)

            let stringIndex = sudoku_string_play.indexOf(".");
            if (stringIndex == -1){
                  alert ("No available hints");
                  return;
            }

            let solutionValue = sudoku_solution[stringIndex];

            updateSudokuString(stringIndex, solutionValue, true);
      }

      //



      // Click item in grid --> higlight grid item, rerender button bar
based on mutability and value, do grid/row/col/num highlighting
      // Press number key --> update grid item, rerender button bar based
on value, do grid/row/col/num highlighting
      // Erase --> update grid item, rerender button bar based on value, do
grid/row/col/num highlighting
```

```
      // Undo --> update grid item, rerender button bar if applicable, do
grid/row/col/num highlighting if applicable

      //

      ////////////////////////////////////////////////

      // Update sudoku string

      function updateSudokuString(stringIndex, value, skipAddToHistory){

            // Replace character in sudoku string . . .
            function getNewSudokuString(stringIndex, value){
                  let str = sudoku_string_play + ""
                  if (stringIndex > str.length - 1) return str;
            return str.substring(0, stringIndex) + value +
str.substring(stringIndex + 1);
            }

            let updatedSudokuString = getNewSudokuString(stringIndex,
value);

            if (!skipAddToHistory){
                  history.push({
                        stringIndex : stringIndex,
                        previousValue : sudoku_string_play[stringIndex],
                        newValue : updatedSudokuString[stringIndex]
                  })
                  updateHistory(
                        history
                  )
            }

            setSudokuString(updatedSudokuString);

      }

      // Item pressed in grid
            // Highlight grid item
            // Rerender button bar based on mutability and value
            // Do grid/row/col/num highlighting

      useEffect(() => {

            updateNumberButtons();

      }, [selectedGridItem]);

      // Number button pressed
            // Update grid item

      const handleNumberPressed = (number) => {

            if (selectedGridItem == undefined) return;

            let stringIndex = selectedGridItem.stringIndex;
```

```
            if (sudoku_string_master[stringIndex] !== ".") return; // Item
is immutable

            // If the current value is the button pressed, clear the value
            // Otherwise, set to pressed value

            updateSudokuString(stringIndex,
(sudoku_string_play[stringIndex] == number) ? "." : number);

      }

      // Sudoku cell updated . . .
            // Check for completion
            // Rerender button bar based on value
            // Do grid/row/col/num highlighting

      useEffect(() => {

            if (sudoku_string_play.indexOf(".") == -1){
                  // sudoku has been filled in!
                  // compare to solution . . .
                  if (sudoku_string_play == sudoku_solution){
                        alert("You solved it!")
                  } else {
                        alert("Not a valid solution . . .")
                  }
            }

            updateNumberButtons();

      }, [sudoku_string_play]);


      ////////////////////////////////////////////////

      const Grid = ({gridID}) => (
            <View style={[gameStyles.grid, gameStyles[`grid_${gridID}`]]}>
                  {
                        [
                              `${gridID}_i1`, `${gridID}_i2`,
`${gridID}_i3`,
                              `${gridID}_i4`, `${gridID}_i5`,
`${gridID}_i6`,
                              `${gridID}_i7`, `${gridID}_i8`,
`${gridID}_i9`,
                        ].map(function(internalGridItemID){

                              return GridItem({
                                    gridID: gridID,
                                    indexObject :
indexTracker.getTracker({gridItem : internalGridItemID})
                              })

                        })
                  }
            </View>
      )
```

```
      const GridItem = ({gridID, indexObject}) => (
            <View style={gameStyles.gridItemContainer}>
                  <Pressable

                        activeOpacity={1}
                        underlayColor={'#e6e6e6'}

                        style={[
                              gameStyles.gridItemHighlight,
                              (selectedGridItem &&
(selectedGridItem.table.row == indexObject.table.row ||
selectedGridItem.table.col == indexObject.table.col)) ? {backgroundColor :
"#e9e9e9"} : null, // Same row/column
                              (selectedGridItem &&
selectedGridItem.gridItem == indexObject.gridItem) ? {backgroundColor :
"#d4d4d4"} : null, // Currently selected grid item
                              (selectedGridItem &&
sudoku_string_play[selectedGridItem.stringIndex] !== "." &&
selectedGridItem.gridItem !== indexObject.gridItem &&
(selectedGridItem.table.row == indexObject.table.row ||
selectedGridItem.table.col == indexObject.table.col) &&
sudoku_string_play[selectedGridItem.stringIndex] ==
sudoku_string_play[indexObject.stringIndex]) ? {backgroundColor: "#ff9999"}
: null, // bad duplicate
                        ]}

                        onPressIn={() => {
                              setSelectedGridItem(indexObject)
                        }}
                  >

                        <Text style={[
                              gameStyles.gridItemText,

(sudoku_string_master[indexObject.stringIndex] == ".") ? {color :
"#0f48a2"} : null, // default text color of a mutable item
                              (selectedGridItem &&
sudoku_string_play[selectedGridItem.stringIndex] !== "." &&
sudoku_string_play[selectedGridItem.stringIndex] ==
sudoku_string_play[indexObject.stringIndex]) ? {fontWeight : "bold"} :
null, // Similar values
                        ]}>

{`${sudoku_string_play[indexObject.stringIndex] == "." ? "" :
sudoku_string_play[indexObject.stringIndex]}`}
                        </Text>
                  </Pressable>
            </View>
      )

      const NumberButton = ({buttonNumber}) => (

            <TouchableHighlight
                  activeOpacity={1}
                  underlayColor={'#e9e9e9'}
                  style={gameStyles.numberButton}
```

```
                  onPress={() => {
                        handleNumberPressed(buttonNumber.replace("b", ""));
                  }}
            >

                  <Text style={[gameStyles.numberButtonText,
gameStyles["numberButtonText_" + buttonHiglights[buttonNumber]]]}>
                        {buttonNumber.replace("b", "")}
                  </Text>
            </TouchableHighlight>
      )



      //

      return (

            <View style={styles.container}>

                  <StatusBar style="auto" />

                  <View style={gameStyles.gameScreen}>

                        <View style={gameStyles.topbar}>

                              <TouchableHighlight
                                    style={gameStyles.backButton}
                                    activeOpacity={1}
                                    underlayColor={'#d4d4d4'}
                                    onPress={() => {
                                          navigation.navigate('Levels')
                                    }}
                              >

                                    <View
style={gameStyles.back_button_view}>
                                          <Image
                                                style =
{gameStyles.back_button_img}

source={require("./../assets/back.png")}
                                          >

                                          </Image>

                                          <Text
style={gameStyles.backButtonText}>
                                                Back
                                          </Text>

                                    </View>


                              </TouchableHighlight>

                              <View style={gameStyles.topbar_right}>
```

```
                                    <TouchableHighlight
style={gameStyles.topbar_right_button}
                                          activeOpacity={1}
                                          underlayColor={'#d4d4d4'}
                                          onPress={() => {
                                                alert('Open Themes Menu')
                                          }}
                                    >

                                          <View>
                                                <Image
                                                      style =
{gameStyles.topbar_img_icon}
source={require("./../assets/palette.png")}
                                                >

                                                </Image>
                                          </View>

                                    </TouchableHighlight>

                                    <TouchableHighlight
style={gameStyles.topbar_right_button}
                                          activeOpacity={1}
                                          underlayColor={'#d4d4d4'}
                                          onPress={() => {
                                                alert('Open Settings Menu')
                                          }}
                                    >

                                          <View>
                                                <Image
                                                      style =
{gameStyles.topbar_img_icon}
source={require("./../assets/settings.png")}
                                                >

                                                </Image>
                                          </View>

                                    </TouchableHighlight>

                              </View>

                        </View>

                        <View style={gameStyles.gameContainer}>
                              <View style={gameStyles.gameHeader}>
                                    <Text
style={gameStyles.difficultyText}>{difficulty.toUpperCase()}</Text>
                                    <Text>{`LEVEL ${level_no}`}</Text>
                              </View>
```

```
                        <View style={gameStyles.timerHeader}>
                              <Text>10:00</Text>
                        </View>


                        {/* Main Sudoku Block*/}

                        <View style={gameStyles.sudokuContainer}>

                              {/* Spawn Subgrids */}
                              {
                                    [
                                          "top_left", "top_middle",
"top_right",
                                          "center_left",
"center_middle", "center_right",
                                          "bottom_left",
"bottom_middle", "bottom_right",
                                    ].map(function(gridID){
                                          return Grid({gridID :
gridID});
                                    })
                              }
                        </View>

                  </View>

                  <View style={gameStyles.buttonContainer}>

                        <View style={gameStyles.functionContainer}>

                              <TouchableHighlight
                                    style={gameStyles.function_button}
                                    activeOpacity={1}
                                    underlayColor={'#e9e9e9'}
                                    onPress={handleUndoPressed}
                              >

                                    <View
style={gameStyles.function_button_view}>
                                          <Image
                                                style =
{gameStyles.function_button_img}

source={require("./../assets/undo.png")}
                                          >

                                          </Image>

                                          <Text
style={gameStyles.function_button_label}>Undo</Text>
                                    </View>

                              </TouchableHighlight>

                              <TouchableHighlight
```

```
                                    style={gameStyles.function_button}
                                    activeOpacity={1}
                                    underlayColor={'#e9e9e9'}
                                    onPress={handleErasePressed}
                              >

                                    <View
style={gameStyles.function_button_view}>
                                          <Image
                                                style =
{gameStyles.function_button_img}

source={require("./../assets/eraser.png")}
                                          >

                                          </Image>

                                          <Text
style={gameStyles.function_button_label}>Erase</Text>
                                    </View>

                              </TouchableHighlight>

                              <TouchableHighlight
                                    style={gameStyles.function_button}
                                    activeOpacity={1}
                                    underlayColor={'#e9e9e9'}
                                    onPress={handleHintPressed}
                              >

                                    <View
style={gameStyles.function_button_view}>
                                          <Image
                                                style =
{gameStyles.function_button_img}

source={require("./../assets/hint.png")}
                                          >

                                          </Image>

                                          <Text
style={gameStyles.function_button_label}>Hint</Text>
                                    </View>

                              </TouchableHighlight>

                        </View>

                        <View style={gameStyles.numbersContainer}>
                              {
                                    [
                                          "b1", "b2", "b3",
                                          "b4", "b5", "b6",
                                          "b7", "b8", "b9",
                                    ].map(function(buttonNumber){
                                          return
```

```
NumberButton({buttonNumber : buttonNumber})
                                                })
                                        }
                                </View>

                        </View>

                </View>

        </View>
    );
}


// STYLINGS

const gridThemeConfig = {
    sudokuGridBorderWidth : 2,
    sudokuGridBorderColor : "#efefef",
}

const styles = StyleSheet.create({

    container: {
        flex: 1,
        backgroundColor: '#fff',
    },

    // MENU PAGE
    menuContainer : {
        flex: 1,
        backgroundColor: '#fff',
        padding : 40,
    },

        // MENU LOGO

    logoView : {
        flex : 2,
        alignItems : "center",
        justifyContent : "center"
    },

    logoTop : {
        alignItems : "center",
        justifyContent : "center",
        textAlign : "center",
        backgroundColor : "#007acc",
        borderRadius : 20,
        width : 120,
        height : 120
    },

    logoTopText : {
        textAlign : "center",
        fontSize : 74,
        color : "#fff",
```

```
      fontWeight : "bold"
},

logoBottom : {
      fontSize : 32,
      fontWeight : "100"
},



      // MENU LIST & BUTTONS

menuListView : {
      flex : 3
},

menuButton : {
      alignContent : "center",
      justifyContent : "center",
      backgroundColor : "#007acc",
      height : 80,
      paddingLeft : 24,
      borderRadius : 4,
      marginBottom : 10
},

menuButtonText : {
      color : "#fff",
      fontSize : 22
},

menuButtonSubtext : {
      color : "#fff",
      fontSize : 12,
      position : "absolute",
      textAlign : "right",
      right : 10,
      bottom : -18
},

// LEVELS PAGE

levelsContainer : {
      flex: 1,
      backgroundColor: '#fff',
      padding : 20,
},

      // LEVELS HEADER
levelsHeader : {
      flex : 1,
      alignItems : "center",
      justifyContent : "center",
},

levelsHeaderTop : {
      fontSize : 42
```

```
	},

	levelsHeaderBottom : {
		fontSize : 24
	},

		// LEVELS LIST & BUTTONS

	levelsListView : {
		flex : 3
	},

	levelButton : {
		alignContent : "center",
		justifyContent : "center",
		backgroundColor : "#007acc",
		height : 60,
		paddingLeft : 20,
		borderRadius : 2,
		marginBottom : 5
	},

	levelButtonText : {
		color : "#fff",
		fontSize : 18
	},

	levelButtonSubtext : {
		color : "#fff",
		fontSize : 12,
		position : "absolute",
		textAlign : "right",
		right : 6,
		bottom : -12
	},

		// LEVELS BACK BUTTON

	levelsBackButton  :{
		marginTop : 40,
		width : 62,
		height : 62,
		borderRadius : 50,
		backgroundColor : "#007acc",
		alignItems : "center",
		justifyContent : "center",
		padding : 10,
	},

	levelsBackButtonIcon : {
		width : 32,
		height : 32
	}
})
```

```
const gameStyles = StyleSheet.create({

      gameScreen : {
            flex : 1,
            alignItems: 'center',
            justifyContent: 'space-between',
            padding : 20,
            paddingTop : 40,
      },

      // INNER CONTAINERS

      topbar : {
            marginTop : 10,
            height : 38,
            width : "100%",
            flexDirection : "row",
            justifyContent : "space-between",
            alignSelf : "flex-start",
      },

      gameContainer : {
            flex : 8,
            padding : 10,
            width : "100%",
            alignItems : "center",
            justifyContent : "center",
      },

      buttonContainer : {
            flex : 3,
            padding : 10,
            width : "100%",
            alignContent : "center",
            justifyContent : "center",
            marginBottom : 40
      },

      // TOPBAR SUBCONTAINER

            // BACK BUTTON

      backButton : {
            borderWidth : 0,
            borderRadius : 4,
            borderColor : "#444",
            alignItems : "center",
            justifyContent : "center",
            paddingLeft : 4,
            paddingRight : 4
      },

      back_button_view : {
            alignItems : "center",
            justifyContent : "space-between",
            flexDirection : "row"
      },
```

```
back_button_img : {
      height : 22,
      width : 22,
      marginRight : 8
},

backButtonText : {
      color : "#444",
      textAlign : "right"
},

topbar_right : {
      flexDirection : "row",
      alignItems : "flex-end",
      justifyContent : "flex-end",
},

      // THEME BUTTON
      // SETTINGS BUTTON

topbar_right_button : {
      height : "100%",
      width : 40,
      marginLeft: 5,
      alignItems : "center",
      justifyContent : "center",
      borderRadius : 4,
},

topbar_img_icon : {
      height : 22,
      width : 22
},

// GAME SUBCONTAINER

      // HEADER

gameHeader : {
      alignItems : "center",
      marginBottom : 10
},

difficultyText : {
      color : "#6687b8",
      fontSize : 26,
},

      // TIMER

timerHeader : {
      width : "100%",
      alignItems : "flex-end",
      color : "#eaeaea",
      paddingRight : 4,
      marginBottom : 5
```

```
    },

        // GAME SQUARE CONTAINER

    sudokuContainer : {
        width : "100%",
        aspectRatio : 1,
        alignItems : "center",
        justifyContent : "space-between",
        flexDirection : "row",
        flexWrap : "wrap"
    },

    grid : {
        width : "33.33%",
        height : "33.33%",
        alignSelf : "flex-start",
        alignItems : "center",
        justifyContent : "space-between",
        flexDirection : "row",
        flexWrap : "wrap",
        padding : 2,
    },

    gridItemContainer : {
        width : "33.33%",
        height : "33.33%",
        alignSelf : "flex-start",
        alignItems : "center",
        justifyContent : "center",
        padding : 2,
    },

    gridItemHighlight : {
        width : "100%",
        height : "100%",
        alignItems : "center",
        justifyContent : "center",
        borderRadius : 4,
        backgroundColor : "#f9f9f9",
    },

    gridItemText : {
        color : "#000000"
    },

    // BUTTON SUBCONTAINER

        // GAME BUTTONS

    functionContainer : {
        width : "100%",
        marginTop : 30,
        flexDirection : "row",
        justifyContent : "space-around",
        alignContent : "center"
    },
```

```
function_button : {
      padding : 10,
      borderRadius : 4,
},

function_button_view : {
      alignItems : "center",
      justifyContent : "center",
},

function_button_img : {
      height : 32,
      width : 32
},

function_button_label : {
      marginTop : 4,
      fontSize : 12
},

      // NUMBER BUTTONS

numbersContainer : {
      width : "100%",
      marginTop : 20,
      flexDirection : "row",
      justifyContent : "space-around",
      alignContent : "center"
},

numberButton : {
      flex : 1,
      borderRadius : 4
},

numberButtonText : {
      fontSize: 34,
      textAlign : "center"
},
numberButtonText_active : {
      color : "#6687b8",
      fontWeight : "100"
},
numberButtonText_enabled : {
      color : "#000",
},
numberButtonText_disabled : {
      color : "#c5c5c5",
},


// GRID STYLINGS

grid_top_middle : {
      borderLeftWidth : gridThemeConfig.sudokuGridBorderWidth,
      borderRightWidth : gridThemeConfig.sudokuGridBorderWidth,
```

```
            borderColor : gridThemeConfig.sudokuGridBorderColor
      },
      grid_bottom_middle : {
            borderLeftWidth : gridThemeConfig.sudokuGridBorderWidth,
            borderRightWidth : gridThemeConfig.sudokuGridBorderWidth,
            borderColor : gridThemeConfig.sudokuGridBorderColor
      },

      grid_center_left : {
            borderTopWidth : gridThemeConfig.sudokuGridBorderWidth,
            borderBottomWidth : gridThemeConfig.sudokuGridBorderWidth,
            borderColor : gridThemeConfig.sudokuGridBorderColor
      },
      grid_center_right : {
            borderTopWidth : gridThemeConfig.sudokuGridBorderWidth,
            borderBottomWidth : gridThemeConfig.sudokuGridBorderWidth,
            borderColor : gridThemeConfig.sudokuGridBorderColor
      },

      grid_center_middle : {
            borderLeftWidth : gridThemeConfig.sudokuGridBorderWidth,
            borderRightWidth : gridThemeConfig.sudokuGridBorderWidth,
            borderTopWidth : gridThemeConfig.sudokuGridBorderWidth,
            borderBottomWidth : gridThemeConfig.sudokuGridBorderWidth,
            borderColor : gridThemeConfig.sudokuGridBorderColor,
      }
});

export default GameScreen;
```

game_levels.json

```
{
      "easy": {
            "1":
"51469873....24.1566.2.5.4892.746..9.391825674..6..9.2.123.84965745936218968512347",
            "2":
"973481.2.52693784114.652..97912.8...3647159822853.9.174.9176258812594.....8.3194",
            "3":
"481527.96932.6.578.769.3241625..94..71..5.6.9..463.7.5268..5934359248167147396852",
            "4":
"..29368744.6..21353.845.962769.24381.831.74561.53.8.296247135...31685.478574961.",
            "5":
"5..283.711.79.5.2..821.7.954.832.519..3579846956418237...8947528257319647..652183",
            "6":
"7142.569.263891475895764132372189.641596..283.865.3719.31......647358..1.28.163.7",
            "7":
"3284.69756915872434.529.68178.6253..5398417..2163798549.715243814...85...52.3.1..",
```

```
            "8":
"524.96813.963.4.5....85.4967..9351649.3681.27165427389471569238658 2.39.1..
91.86.5",
            "9":
"5.421.983.3..8.5166185934..425178.397..432851183.65.4.84732916.2..85739.3.
9641278",
            "10":
"9.815327.1.76..9.53.5....8.281.764935743..62169.4217587.281.36943926581781
.937542",
            "11":
"...241385.21.3.6...386571921.4.825.985..647213...958.628547691374691.25891
3528467",
            "12":
".59237648.275.839138619472571.856932638729..4.954138678.29.5..35.36.1..99.
13.2..6",
            "13":
"971385624.42.6.35.563.4.98....7214966974382151.4596..32196547384...7316.73
681.54.",
            "14":
"2671.85.4485..72.139145.876178..9345946583.1.532714.6861.8754.9754....8382
934..57",
            "15":
"6738.915.49156387..8547.9639523847161.6.9.5488.7.56...524.18..736..45.9171
9632485",
            "16":
"172.349..584169.23693572.1..2864153936975.241451923867836.9517494531..8.21
.......",
            "17":
"179..86.53645.1728258.4..1342.9835.181.45239.593167.847456198326.2.741599.
1..54..",
            "18":
"765.81..4.8.794.5194.526738426.57.93319642.878.7.132.65941783621..269475.7
...5819",
            "19":
"795.23184183457.9.4629.17538475623.935.19842.9217348.5.1934..78...27..41.7
48.953.",
            "20":
"4..1937657.5264...3.68754.29.1352847247618539853749.2...948625363.5279..5.
.93167.",
            "21":
"71.92.683.98..3127632178594357896241...217935921...87.2..7.1459.4...971217
9452368",
            "22":
"1.....479......1.24.271.5.826157894387934125634596278168412 7.9592783561.51
3694..7",
            "23":
"65.437..848.652.3737.819465.17.6..83938.71.46.46.837.186419537279532..1.1.
3746859",
            "24":
"4579261836918375..283541769.6.......748293651.1.6.....13645289797436821582
5..9436",
            "25":
"37261598.68923.715415..7.367981.6.4.5463298712.1.486.99.3471528.5786249.82
4.9....",
            "26":
"846395712.572613..13247895636894257129451....57183642962.1548.7.15.8.2...8
..2.1.5",
```

```
            "27":
"25764931.8.951327..132785.9.28..71.5.7.164832..185279.1967254837.24.69515.
4..1627",
            "28":
"..592..4..268547....461.52.593761284.784923152.13859674621.9853.5.248.9.81
9536472",
            "29":
"...23.185.3.75.9.425..8.63.9.254371.516872.937436918528214653..365..724849
7328561",
            "30":
"69.53.71878.91.3..5318762941..3.74..4762.19..35.46.172.631..54781574362994
7625831",
            "31":
"4.2519.7.1...7...3.79236.45584.9.2.69216857..7631245.93479518..21586349769
8742351",
            "32":
"27948316546127593885391.427598.2731.61..3.7.2732.416.9.2739854638..5429194
.......",
            "33":
"...14...32438697151..23..469..4765.1....51..44513286978967143525246831793
7592468",
            "34":
"....746159.4.518725178.2394....9652725.738946679...1831354.726.79261345848
6..9731",
            "35":
"38962574115..842.6642...5.842..639.771...93629..2..8.4571432689.3479612529
6158473",
            "36":
"6917285..2539417867483561.25.28.9...1.92.485.8.45.7.2.32.185..941569237898
647.215",
            "37":
".691....8.816.3.9.537928164.728..65364573.8193985164728263...417.348192691
426.38.",
            "38":
"16952.34874.98.62585243.197621.93.8.9..15426.5342...19285.19.764168759...9
7642.51",
            "39":
"786.9..2531257689449.2.873.12468..73857..146.639..7.81...7.93.857384261996
8153247",
            "40":
"97185.263835296714426371..5594..8327613527...78294365116843957224...5.3.35
...2..."
      },
      "medium": {
            "1":
"179543628582.7..43643.2..7.85439271.361.8..92927.1.38...8.5..6..96.3.25..1
5.6.83.",
            "2":
"4...718323872594.62.6483597864.9215352314.769.7.365.4.7.2.............4
8537921",
            "3":
"96721.8..8327.916.541863729.9.3875123254916.7718526.9..5...82...8.67...1.7
......8",
            "4":
"3..1.945.5.....2.4.15.239.65382.914...9.1.63..465.287746.158.91..7386.583
5496172",
            "5":
```

```
".1.2......7.8.19...825..61....7..18...81..7.97913.84651394825768269573414576613298",
            "6":
"1.67.3...3.51..67.7.4526...54736..2.9628547...13279...278435...651982347439617...",
            "7":
".1.689753589.43261..72514.915.492876.4...5392...3.814....51.93...183.5..43592..18",
            "8":
"5431687296782395149125476387569..3.2329....61481623957..4.....5..7.....6..5.....3",
            "9":
"783..64.25..3..1866..528379.3.8529678..469.132..73154.....83.253..647891..82.5.34",
            "10":
"4981..2366239845717..2.6948.5..41.2..3476215927139568...7.29...1.957..62..2.1....",
            "11":
"..27.834...81.47...4.5392.837649.821.8.2.36..2.48.69.38653.7.9....9.1486491682537",
            "12":
"6..541.87853976.12.7.8235691...98.5..6..15...587...9.1248..7.95..62598747951.46.3",
            "13":
"526..1..88..6542.141.8..6.5.38..57..941267853.75..8....52746189789...5..164589327",
            "14":
"...4.328.834.7196.912.58743...7....4..31.4..8.4...9..7421365879659847312387912 4..",
            "15":
"...37..5.3..1..76....2..83...34.25966549873.2..1635478...76318513852.6.7567841923",
            "16":
".38..62412.6.48759749251638..24.31.6364.128.59716.53246..8.4.1.1..56.48.4.....6.",
            "17":
"879.4.1263.428.759..597.4835.26.8..7..1.9463..9673.518643857.9..5712.86412.......",
            "18":
"472635...1..784625.5.291473.2.418756561372894784956231...1.75....58....7..75.....",
            "19":
"6125.9738745283.69938...5423246189571..395284589.2.6.3491852376..............",
            "20":
"36.84.2..41.932..828.67.3..57.2638..9...87...8...19...693754182152398476748126 9..",
            "21":
"53.6..4.....34.5...74589613...42537949273185635.96...1...274.35.2.156.84.45893.62",
            "22":
"45.1968278624739157192853.3.69.....8.578....6.32...8.7.23.....9898..2.34.146839752",
            "23":
"..7.6...5.365...27.1527...8762158...381..765254932..71.7.632589253.8.7166987.52..",
            "24":
```

```
"..3...765...5..839.56.3.42..81.7.546.6....978794.653124386..157619.5.28357
2183.94",
            "25":
".1.74..23.3.1.....742653198493816..2581372..9276594831..9...3....7.3.9..36
4987215",
            "26":
".861.59323.5.964...913.28..8792.43..2537.86.9.14953..8..84271931.26.95..9.
75.12..",
            "27":
"47.9..1235.32...941..634..8692....51....9273635.14.2897.84239152..85.36793
5761...",
            "28":
"179.253865.4...2918..391475..2.13.6...3......6.1...8329175..62334867.1.926
5139748",
            "29":
"51834.67.2691573484.768..1.3..9.8.6.8..5.1.3..2.4.3.8.6.281.793...7..82678
3296451",
            "30":
"694538...285719436173264598462173985819425673537896241.4.6.............
.......",
            "31":
".24..78....8..4927.7..8.4......7.3.443..91278..743.5.164912378518274563.75
3869142",
            "32":
"..1573.8.4.3.8.72..87....3....248173712635894348791562 8743.2......4.73.8.3
586924.",
            "33":
".9..712...2..459.7.7..291..263987.14.4..62.9.957.34..2735.1.629689253471.1
27963..",
            "34":
"31826.754.79..5...56.741.8.9..6.7..8..7..4.6...6..3.4768193247529547683174
3..8692",
            "35":
"...67.98.8.6594..7974218635...932.61619.853......61..9.6..49728....5749649
7826153",
            "36":
"6.931..45813542769.4..69.3129713548638629451715 46..9239.......246...31.8..
......4",
            "37":
"72981463..3.297..88.4635972......3211.35.2769..21735842..7..1.33.1.2.8.747
835.2..",
            "38":
"...75.94847.19.3.65..46.721...32..747..6..2832348.5.69.45937..29...8643736
721.895",
            "39":
"5.94....8.6.9...542345.819..4..25.....5.498.2.2.781.45.8.1345.939285746145
1296783",
            "40":
"743869521.18..53.7.52731..8.6..7.....7.......824516.732913..7565.7692..448
6157239"
      },
      "hard": {
            "1":
".456.1.7..2.9...1...98...5..7358419645619328798176..43.3..5.....9247.8....
..1..2.",
            "2":
"..84.2.71.7.1.3.68.1...8.2948.6..13212...46956...21.478.123..5.79.84.21...
```

```
..1..8.",
            "3":
"6.43.15723..54.86995268.431..81..65..9.4..72.7.32.8..........583...524.5.
..243.6",
            "4":
"2..45..9.41529.73....17.425...94..6.924.8..7.....2.94.6.2819.57....64289..
..32614",
            "5":
".293..4.........277.62.1...9756241.83.2......4..1.3.72684.1..9319743285625
3...7..",
            "6":
".59816743...529..886....59261.....75.931...24.8.........5.42.869486512.7.2
6...45.",
            "7":
"7.3....49.6153.28.95.4..3.6..7...8.554...37.....7.54.3634251978...37.6.227
569....",
            "8":
".27..4913.95.73...63..2.5..2.8..6.....63.2..8..3.182...8923......169.83236
2481759",
            "9":
"..796....5297.4...6.15.24972.38.7.4.865.4..7...4.....8..2...78.1.837..6.97
6428351",
            "10":
".5...289.29.8..15.18...927..295364..361784925.45921......2.8..99.2........
8.9.742",
            "11":
"872...416596741.83...682759.....8572..3..76487.....9314.7....95.......6...
.875124",
            "12":
"9.......51.7529.3625..3..79315...792469275318872391..45.....9816.......77.
.......3",
            "13":
".7..36....3..58.7..9.12734..4.7.9....6934.71.75.2.1..4.1...2457..75148.348
5673...",
            "14":
"6971..532..53..94.3.....7.81.....67.28.7..15....95..835.841.39.7..6..8..93
2875416",
            "15":
".635724.85....6732274.8.6.5..7...95..9.7.584.8.56..3...5..6718.7....156..3
..5.279",
            "16":
"...26.5...6.74.....2..5....15..8.....48512...293476158...124....1.83742547
2695381",
            "17":
"..5.....9..7.59....1.3.6587583....42.......5.....35.78.5.973814398614725..
1582396",
            "18":
".5.249.17..2516.9...9783...9.5867.....4921576.6.354.2....198..2....72...29
..35.81",
            "19":
"1...82.595...3....7..15....9..37..6.6...41.9545..96...3..527.48.4591327627
9468..1",
            "20":
"....1..5.58..4...9....85.....6578.4..7.4935.6.5.261...893157462...82937572
5634...",
            "21":
"....1........5........3....865124.7...3.87.24742.93.81628341.9.35927814.47
```

```
1965.3.",
            "22":
"...891...9276..381418732695..5917423792463..8341528.6...4.......7...6.....
.1.....",
            "23":
"5..6.94..96..5.2..1.....5966589...2.3.98.61..7.1.3.8694...6.91.81579364229
6......",
            "24":
"..7.28....25.1..8783.75946227..83...38..7.2..51..92873.62931..8.5384762...
........",
            "25":
"74.5..16..6..1.3...1..76..8.51.67...68..249...2...5..6132698....9.7516..57
6243891",
            "26":
"....2.........2........5...9.14.837...973..53.72.54..15.83274664875931272
3461958",
            "27":
"627419835153286...849753..1..8..4..6.968.......2965.8..856...7.971..8..2..
4.....8",
            "28":
"973..12..65.....932...3976.3.6.4....4..39..767951.63.2..791..28142.5..3.83
9.24...",
            "29":
".5.17..947.984.351.1.5..7.6.3.2....7....17..3.7......9..73.5412...7.493534
5921678",
            "30":
".......8.....7..6......9.3.2317948569681527437.4386912.72..5.9839.2.8.7...
.9.7.2.",
            "31":
"714.85.362.81..47.9.3.4..811..468...6.935...443...16.88..6.41..5418...6...
651.8..",
            "32":
"834..5.9..7...3..416...4...42.38..1..........6........398652471241978563756
431289",
            "33":
".57..9314..8..5692.39...758584..3167792...583361...249..31.6.259....24...2
.......",
            "34":
"..2.1.5..153298.....6..721.6417.9.2.385162...927.8.16..6892...121487..5..3
......2",
            "35":
"948.5.2361.7693.4.......971......12...1...76.2...1638.8.627.453..4.38612..
2.6.897",
            "36":
"...4781.61876325494..5197.8...725.1....361.955.1894.7..1.2........18..5...
.9..261",
            "37":
"745931.68..6857.4....42635...4...576..378541.5.76.4...47.56........4.7.5.5
..7.624",
            "38":
"5..9.3864...478215..45.6739.1.23.....4.165..7.3.....52.6.35294..5....62.42
.6..573",
            "39":
"..3..425.2......4..6527..91.36.8.412.2.14..38148....696923.71848..4...73.7
4....25",
            "40":
"5.1...738...3.8459.4..75612289.37164436.9.8.517....92376.82.59.........7..
```

```
.7....6"
      },
      "insane": {
            "1":
".....79.29.71...4682.4..375.9....754..8...26.................7........1.7
....4.8",
            "2":
"..32.....7.65..1.....8......1.7........1.4.5....3........671..896.45.7.2.4
79.....",
            "3":
".........98.5.24..1..4.....2.......9...75............357..8.2.1895..3..24
639....",
            "4":
"6...8421......5873.38....46..5..7..2.23........1............6.....1.......
..59381",
            "5":
"........6....96........3....5.......3.2.6..5.7..2.5.........61.637.5.24912
46.9...",
            "6":
"...416.....6239...2..5786.9..8.........985...1...6....3.2.5..6...18.3.....
.........",
            "7":
"67..9....18...6.5.92..1......1......796.2..3.2.85..7......7..2..6..5.....1
.....9.",
            "8":
"..7193.....37...95..845.3....1....2..65.49....8..1....8..92.1.3...........
...1...",
            "9":
"91.......25.....9....23..1....8..1.7........6.8.37.9...3149..78.....3..1..
....3.9",
            "10":
".3......1.125.........1...4...2..81.12...73458..1.4.....1.2........9....2.
....198",
            "11":
".........5...8....3.........5.26......357........3.57..1..2..54465.1....23
8945...",
            "12":
"4.....9..6...4.83..........9643..1..8751..3462..6.45.9........4...4.......
.......3",
            "13":
"79.......2.4.........2.....3...................2..5......682......95328782
5471369",
            "14":
".3..4.1..4......2.8.71...4.9834.....546.7..91.7.....54.2....4.............
.....19",
            "15":
".4...6.....63..4...3.....69.692..571.2.61.3.8.18.....66.34...1.............
........",
            "16":
"......2............8....3....2..7..3..5....21...6239....1379864..4..1.3..7
....19.",
            "17":
"..8619437314758269967..4.........3.......68.1...........6.................
.......2",
            "18":
".5....6..1...64....6..57....385..7.....38.2..9..74.......1.8......4.....79
16..4..",
```

```
            "19":
"14.298.36.9.361..2...45...1...6.5........261......3....541....7.........
...9...",
            "20":
".76.......84...3..5.....7........476742....39...7...826....2.....9..62..42
....6..",
            "21":
"..6....722.7........1.7.....1......9....5.....4...1..8...8..49...4......17
8594236",
            "22":
"2...6.......8....2.........692....5.....42..9..1...2.3..9....8.824591....5
...892.",
            "23":
"...1.............185......3........7...1.9......28..4.92.51736...6....167
1..385.",
            "24":
".96184...38.5..4.1154....8.....36...8.5........3.5.....4.91.....7.....9..3
........",
            "25":
".8917.36...7.3.9.8..4.89.75..6....2...2..3..7....2.....2.9...8....31.....
.........",
            "26":
"856.9.3...9...6584.4...86.9...2.9..8...67..9.9..........9..4.6.....3.....
....8..",
            "27":
"7.8..2...3.....1.2..2.3.....7....813.4...3.....3....94.............1..2.63
.28745.",
            "28":
"..28..6....6.2.98.584........7.......58....2...3...1..471....9...5.......
9.4576.",
            "29":
"..9.1..25.....2....2.......37.....414..7...92....4...6......2.8.....8463..
..2.179",
            "30":
"1...7...48.......949.....73.1.....9.....2.43.9.6....8.6....13..3.........
1.6.947",
            "31":
"..86....5.1.5.....725..8..6...2....12.........3..7..2...37...82.8.4.....5.
238....",
            "32":
"....3....35..82147..7...3....521.4....3....21..2.498...7....6..5.1...7....
........",
            "33":
"42.....6.8..2............5..12.4..9.............5...7238...97....37942894
7......",
            "34":
"627..9..........5....1.7.64...96.....867.1...........6.9.8..6.3.6.........
5692.1.",
            "35":
"293.57..8.1.92..35....3..29..8.....4..146.9....9...5..1..3...9.9..........
........",
            "36":
"....96..4..24.86....6.......68...3599136.....5..3.9.6........8.8........6.
.8....2",
            "37":
"5.7......8192.67..4.6......162..3.747456.....983..75........1.............
........",
```

```
            "38":
"4..5...2..96..2.4........9..5....7...6........7..8.4...279.18343..4.7....4
.....7.",
            "39":
"7...58...8..7.....53......81583....42479..5...6........2........7........8
...7419",
            "40":
"2...1....9.......3....5.........1.3........71.......9.......38939.184..775
89..146"
      }
}
```

indexTracker.js

```
let indexTracker = {

	////////////////////////////////////
	////////////////////////////////////

	stringIndexToGridItem_Table : {}, // eg. "top_left_i2" --> 1
	gridItemToStringIndex_Table : {},  // eg. 27 --> "middle_left_i1"

	getStringIndexFromGridItem : function(gridItemID){
		return indexTracker.gridItemToStringIndex_Table[gridItemID];
	},

	getGridItemFromStringIndex : function(stringIndex){
		return indexTracker.stringIndexToGridItem_Table[stringIndex];
	},

	////////////////////////////////////
	////////////////////////////////////

	// Inputs:
		// stringIndex (num) eg. 23
		// gridItem (string) eg. top_left_i2
		// table (object) eg. {row : <num>, col : <num>}

	getTracker : function(inputs){

		// Smart validation


		// Expects an object without "row" in it

		if ((typeof inputs !== "object") || (typeof inputs == "object"
&& "row" in inputs)){
			let input_validation = {};
			if (typeof inputs == "string"){
				input_validation.gridItem = inputs;
			} else if (!isNaN(parseInt(inputs))){
				input_validation.stringIndex = inputs.toString();
			} else if ("row" in inputs){
				input_validation.table = inputs;
			} else {
				// Some real invalid stuff got passed over huh . .
.
			}
			inputs = input_validation;
		}

		//

		let outputTracker = {
			stringIndex : inputs.stringIndex,
			gridItem : inputs.gridItem,
			table : inputs.table
		}
```

```
            // todo
            // check if the provided inputs have at least one filled

            if (outputTracker.stringIndex == undefined){
                  if (outputTracker.gridItem){
                        // gridItem to stringIndex
                        outputTracker.stringIndex =
indexTracker.getStringIndexFromGridItem(outputTracker.gridItem);
                  } else if (outputTracker.table){
                        // table to stringIndex
                        outputTracker.stringIndex =
(outputTracker.table.row * 9) + outputTracker.table.col;
                  }
            }

            if (outputTracker.gridItem == undefined){
                  // stringIndex to gridItem
                  outputTracker.gridItem =
indexTracker.getGridItemFromStringIndex(outputTracker.stringIndex);
            }

            if (outputTracker.table == undefined){
                  // stringIndex to table
                  outputTracker.table = {
                        row : (outputTracker.stringIndex -
(outputTracker.stringIndex % 9))/9,
                        col : outputTracker.stringIndex % 9
                  }
            }

            return outputTracker;

      },

      ofRow : {},
      ofColumn : {},

}


function generateIndexTrackerHelpers(){
      // Get the string index for the first item in each subgrid
      let indexTracker_origins = {
            "top_left" : (9 * 0),
            "top_middle" : (9 * 0) + 3,
            "top_right" : (9 * 0) + 6,

            "center_left" : (9 * 3),
            "center_middle" : (9 * 3) + 3,
            "center_right" : (9 * 3) + 6,

            "bottom_left" : (9 * 6),
            "bottom_middle" : (9 * 6) + 3,
            "bottom_right" : (9 * 6) + 6,
      }

      // Build string index from grid id table . . .
```

```
      for (let [subgrid, initial_index] of
Object.entries(indexTracker_origins)){

            // count 3, skip 6, count 3, skip 6, count 3
            let current_index = initial_index - 1;
            let skip_count = 0;

            for (let i = 1; i < 10; i++){


indexTracker.gridItemToStringIndex_Table[`${subgrid}_i${i}`] =
++current_index;

                  skip_count++;
                  if (skip_count >= 3 ){
                        skip_count = 0;
                        current_index = current_index + 6;
                  }
            }

      }

      // Invert table for grid id from string index table . . .

      for (let [gridItemID, stringIndex] of
Object.entries(indexTracker.gridItemToStringIndex_Table)){
            indexTracker.stringIndexToGridItem_Table[stringIndex,
gridItemID];
      }

      // Create ofRow and ofColumn
}



generateIndexTrackerHelpers();


export default indexTracker
```

sudokuGenerator.js

Source: https://github.com/robatron/sudoku.js/

```
/*
Sudoku.js
---------

A Sudoku puzzle generator and solver JavaScript library.

Please see the README for more details.
*/


var sudoku = {};  // Global reference to the sudoku library

sudoku.DIGITS = "123456789";    // Allowed sudoku.DIGITS
var ROWS = "ABCDEFGHI";         // Row lables
var COLS = sudoku.DIGITS;       // Column lables
var SQUARES = null;             // Square IDs

var UNITS = null;               // All units (row, column, or box)
var SQUARE_UNITS_MAP = null;    // Squares -> units map
var SQUARE_PEERS_MAP = null;    // Squares -> peers map

var MIN_GIVENS = 17;            // Minimum number of givens
var NR_SQUARES = 81;            // Number of squares

// Define difficulties by how many squares are given to the player in a new
// puzzle.
var DIFFICULTY = {
    "easy":         62,
    "medium":       53,
    "hard":         44,
    "very-hard":    35,
    "insane":       26,
    "inhuman":      17,
};

// Blank character and board representation
sudoku.BLANK_CHAR = '.';
sudoku.BLANK_BOARD =
"....................................................."+
        ".........................";

// Init
//
// -----------------------------------------------------------------------
function initialize(){
    /* Initialize the Sudoku library (invoked after library load)
    */
    SQUARES             = sudoku._cross(ROWS, COLS);
    UNITS               = sudoku._get_all_units(ROWS, COLS);
    SQUARE_UNITS_MAP    = sudoku._get_square_units_map(SQUARES, UNITS);
    SQUARE_PEERS_MAP    = sudoku._get_square_peers_map(SQUARES,
                                SQUARE_UNITS_MAP);
}
```

```
// Generate
//
//------------------------------------------------------------------------
sudoku.generate = function(difficulty, unique){
    /* Generate a new Sudoku puzzle of a particular `difficulty`, e.g.,

        // Generate an "easy" sudoku puzzle
        sudoku.generate("easy");


    Difficulties are as follows, and represent the number of given squares:

            "easy":         61
            "medium":       52
            "hard":         43
            "very-hard":    34
            "insane":       25
            "inhuman":      17


    You may also enter a custom number of squares to be given, e.g.,

        // Generate a new Sudoku puzzle with 60 given squares
        sudoku.generate(60)


    `difficulty` must be a number between 17 and 81 inclusive. If it's
    outside of that range, `difficulty` will be set to the closest bound,
    e.g., 0 -> 17, and 100 -> 81.


    By default, the puzzles are unique, uless you set `unique` to false.
    (Note: Puzzle uniqueness is not yet implemented, so puzzles are *not*
    guaranteed to have unique solutions)

    TODO: Implement puzzle uniqueness
    */

    // If `difficulty` is a string or undefined, convert it to a number or
    // default it to "easy" if undefined.
    if(typeof difficulty === "string" || typeof difficulty ===
"undefined"){
        difficulty = DIFFICULTY[difficulty] || DIFFICULTY.easy;
    }

    // Force difficulty between 17 and 81 inclusive
    difficulty = sudoku._force_range(difficulty, NR_SQUARES + 1,
            MIN_GIVENS);

    // Default unique to true
    unique = unique || true;

    // Get a set of squares and all possible candidates for each square
    var blank_board = "";
    for(var i = 0; i < NR_SQUARES; ++i){
        blank_board += '.';
```

```
    }
    var candidates = sudoku._get_candidates_map(blank_board);

    // For each item in a shuffled list of squares
    var shuffled_squares = sudoku._shuffle(SQUARES);
    for(var si in shuffled_squares){
        var square = shuffled_squares[si];

        // If an assignment of a random chioce causes a contradictoin, give
        // up and try again
        var rand_candidate_idx =
                sudoku._rand_range(candidates[square].length);
        var rand_candidate = candidates[square][rand_candidate_idx];
        if(!sudoku._assign(candidates, square, rand_candidate)){
            break;
        }

        // Make a list of all single candidates
        var single_candidates = [];
        for(var si in SQUARES){
            var square = SQUARES[si];

            if(candidates[square].length == 1){
                single_candidates.push(candidates[square]);
            }
        }

        // If we have at least difficulty, and the unique candidate count
is
        // at least 8, return the puzzle!
        if(single_candidates.length >= difficulty &&
                sudoku._strip_dups(single_candidates).length >= 8){
            var board = "";
            var givens_idxs = [];
            for(var i in SQUARES){
                var square = SQUARES[i];
                if(candidates[square].length == 1){
                    board += candidates[square];
                    givens_idxs.push(i);
                } else {
                    board += sudoku.BLANK_CHAR;
                }
            }

            // If we have more than `difficulty` givens, remove some random
            // givens until we're down to exactly `difficulty`
            var nr_givens = givens_idxs.length;
            if(nr_givens > difficulty){
                givens_idxs = sudoku._shuffle(givens_idxs);
                for(var i = 0; i < nr_givens - difficulty; ++i){
                    var target = parseInt(givens_idxs[i]);
                    board = board.substr(0, target) + sudoku.BLANK_CHAR +
                        board.substr(target + 1);
                }
            }

            // Double check board is solvable
```

```
            // TODO: Make a standalone board checker. Solve is expensive.
            if(sudoku.solve(board)){
                return board;
            }
        }
    }

    // Give up and try a new puzzle
    return sudoku.generate(difficulty);
};

// Solve
//
// ----------------------------------------------------------------------
sudoku.solve = function(board, reverse){
    /* Solve a sudoku puzzle given a sudoku `board`, i.e., an 81-character
    string of sudoku.DIGITS, 1-9, and spaces identified by '.',
representing the
    squares. There must be a minimum of 17 givens. If the given board has
no
    solutions, return false.

    Optionally set `reverse` to solve "backwards", i.e., rotate through the
    possibilities in reverse. Useful for checking if there is more than one
    solution.
    */

    // Assure a valid board
    var report = sudoku.validate_board(board);
    if(report !== true){
        throw report;
    }

    // Check number of givens is at least MIN_GIVENS
    var nr_givens = 0;
    for(var i in board){
        if(board[i] !== sudoku.BLANK_CHAR && sudoku._in(board[i],
sudoku.DIGITS)){
            ++nr_givens;
        }
    }
    if(nr_givens < MIN_GIVENS){
        throw "Too few givens. Minimum givens is " + MIN_GIVENS;
    }

    // Default reverse to false
    reverse = reverse || false;

    var candidates = sudoku._get_candidates_map(board);
    var result = sudoku._search(candidates, reverse);

    if(result){
        var solution = "";
        for(var square in result){
            solution += result[square];
        }
        return solution;
```

```
    }
    return false;
};

sudoku.get_candidates = function(board){
    /* Return all possible candidatees for each square as a grid of
    candidates, returnning `false` if a contradiction is encountered.

    Really just a wrapper for sudoku._get_candidates_map for programmer
    consumption.
    */

    // Assure a valid board
    var report = sudoku.validate_board(board);
    if(report !== true){
        throw report;
    }

    // Get a candidates map
    var candidates_map = sudoku._get_candidates_map(board);

    // If there's an error, return false
    if(!candidates_map){
        return false;
    }

    // Transform candidates map into grid
    var rows = [];
    var cur_row = [];
    var i = 0;
    for(var square in candidates_map){
        var candidates = candidates_map[square];
        cur_row.push(candidates);
        if(i % 9 == 8){
            rows.push(cur_row);
            cur_row = [];
        }
        ++i;
    }
    return rows;
}

sudoku._get_candidates_map = function(board){
    /* Get all possible candidates for each square as a map in the form
    {square: sudoku.DIGITS} using recursive constraint propagation. Return
`false`
    if a contradiction is encountered
    */

    // Assure a valid board
    var report = sudoku.validate_board(board);
    if(report !== true){
        throw report;
    }

    var candidate_map = {};
    var squares_values_map = sudoku._get_square_vals_map(board);
```

```
    // Start by assigning every digit as a candidate to every square
    for(var si in SQUARES){
        candidate_map[SQUARES[si]] = sudoku.DIGITS;
    }

    // For each non-blank square, assign its value in the candidate map and
    // propigate.
    for(var square in squares_values_map){
        var val = squares_values_map[square];

        if(sudoku._in(val, sudoku.DIGITS)){
            var new_candidates = sudoku._assign(candidate_map, square,
val);

            // Fail if we can't assign val to square
            if(!new_candidates){
                return false;
            }
        }
    }

    return candidate_map;
};

sudoku._search = function(candidates, reverse){
    /* Given a map of squares -> candiates, using depth-first search,
    recursively try all possible values until a solution is found, or false
    if no solution exists.
    */

    // Return if error in previous iteration
    if(!candidates){
        return false;
    }

    // Default reverse to false
    reverse = reverse || false;

    // If only one candidate for every square, we've a solved puzzle!
    // Return the candidates map.
    var max_nr_candidates = 0;
    var max_candidates_square = null;
    for(var si in SQUARES){
        var square = SQUARES[si];

        var nr_candidates = candidates[square].length;

        if(nr_candidates > max_nr_candidates){
            max_nr_candidates = nr_candidates;
            max_candidates_square = square;
        }
    }
    if(max_nr_candidates === 1){
        return candidates;
    }
```

```
    // Choose the blank square with the fewest possibilities > 1
    var min_nr_candidates = 10;
    var min_candidates_square = null;
    for(si in SQUARES){
        var square = SQUARES[si];

        var nr_candidates = candidates[square].length;

        if(nr_candidates < min_nr_candidates && nr_candidates > 1){
            min_nr_candidates = nr_candidates;
            min_candidates_square = square;
        }
    }

    // Recursively search through each of the candidates of the square
    // starting with the one with fewest candidates.

    // Rotate through the candidates forwards
    var min_candidates = candidates[min_candidates_square];
    if(!reverse){
        for(var vi in min_candidates){
            var val = min_candidates[vi];

            // TODO: Implement a non-rediculous deep copy function
            var candidates_copy = JSON.parse(JSON.stringify(candidates));
            var candidates_next = sudoku._search(
                sudoku._assign(candidates_copy, min_candidates_square, val)
            );

            if(candidates_next){
                return candidates_next;
            }
        }

    // Rotate through the candidates backwards
    } else {
        for(var vi = min_candidates.length - 1; vi >= 0; --vi){
            var val = min_candidates[vi];

            // TODO: Implement a non-rediculous deep copy function
            var candidates_copy = JSON.parse(JSON.stringify(candidates));
            var candidates_next = sudoku._search(
                sudoku._assign(candidates_copy, min_candidates_square,
val),
                reverse
            );

            if(candidates_next){
                return candidates_next;
            }
        }
    }

    // If we get through all combinations of the square with the fewest
    // candidates without finding an answer, there isn't one. Return false.
    return false;
};
```

```
sudoku._assign = function(candidates, square, val){
    /* Eliminate all values, *except* for `val`, from `candidates` at
    `square` (candidates[square]), and propagate. Return the candidates map
    when finished. If a contradiciton is found, return false.

    WARNING: This will modify the contents of `candidates` directly.
    */

    // Grab a list of canidates without 'val'
    var other_vals = candidates[square].replace(val, "");

    // Loop through all other values and eliminate them from the candidates
    // at the current square, and propigate. If at any point we get a
    // contradiction, return false.
    for(var ovi in other_vals){
        var other_val = other_vals[ovi];

        var candidates_next =
            sudoku._eliminate(candidates, square, other_val);

        if(!candidates_next){
            //console.log("Contradiction found by _eliminate.");
            return false;
        }
    }

    return candidates;
};

sudoku._eliminate = function(candidates, square, val){
    /* Eliminate `val` from `candidates` at `square`, (candidates[square]),
    and propagate when values or places <= 2. Return updated candidates,
    unless a contradiction is detected, in which case, return false.

    WARNING: This will modify the contents of `candidates` directly.
    */

    // If `val` has already been eliminated from candidates[square], return
    // with candidates.
    if(!sudoku._in(val, candidates[square])){
        return candidates;
    }

    // Remove `val` from candidates[square]
    candidates[square] = candidates[square].replace(val, '');

    // If the square has only candidate left, eliminate that value from its
    // peers
    var nr_candidates = candidates[square].length;
    if(nr_candidates === 1){
        var target_val = candidates[square];

        for(var pi in SQUARE_PEERS_MAP[square]){
            var peer = SQUARE_PEERS_MAP[square][pi];

            var candidates_new =
```

```
                    sudoku._eliminate(candidates, peer, target_val);

            if(!candidates_new){
                return false;
            }
        }

    // Otherwise, if the square has no candidates, we have a contradiction.
    // Return false.
    } if(nr_candidates === 0){
        return false;
    }

    // If a unit is reduced to only one place for a value, then assign it
    for(var ui in SQUARE_UNITS_MAP[square]){
        var unit = SQUARE_UNITS_MAP[square][ui];

        var val_places = [];
        for(var si in unit){
            var unit_square = unit[si];
            if(sudoku._in(val, candidates[unit_square])){
                val_places.push(unit_square);
            }
        }

        // If there's no place for this value, we have a contradition!
        // return false
        if(val_places.length === 0){
            return false;

        // Otherwise the value can only be in one place. Assign it there.
        } else if(val_places.length === 1){
            var candidates_new =
                sudoku._assign(candidates, val_places[0], val);

            if(!candidates_new){
                return false;
            }
        }
    }

    return candidates;
};


// Square relationships
//
-----------------------------------------------------------------------
// Squares, and their relationships with values, units, and peers.

sudoku._get_square_vals_map = function(board){
    /* Return a map of squares -> values
    */
    var squares_vals_map = {};

    // Make sure `board` is a string of length 81
    if(board.length != SQUARES.length){
```

```
        throw "Board/squares length mismatch.";

    } else {
        for(var i in SQUARES){
            squares_vals_map[SQUARES[i]] = board[i];
        }
    }

    return squares_vals_map;
};

sudoku._get_square_units_map = function(squares, units){
    /* Return a map of `squares` and their associated units (row, col, box)
    */
    var square_unit_map = {};

    // For every square...
    for(var si in squares){
        var cur_square = squares[si];

        // Maintain a list of the current square's units
        var cur_square_units = [];

        // Look through the units, and see if the current square is in it,
        // and if so, add it to the list of of the square's units.
        for(var ui in units){
            var cur_unit = units[ui];

            if(cur_unit.indexOf(cur_square) !== -1){
                cur_square_units.push(cur_unit);
            }
        }

        // Save the current square and its units to the map
        square_unit_map[cur_square] = cur_square_units;
    }

    return square_unit_map;
};

sudoku._get_square_peers_map = function(squares, units_map){
    /* Return a map of `squares` and their associated peers, i.e., a set of
    other squares in the square's unit.
    */
    var square_peers_map = {};

    // For every square...
    for(var si in squares){
        var cur_square = squares[si];
        var cur_square_units = units_map[cur_square];

        // Maintain list of the current square's peers
        var cur_square_peers = [];

        // Look through the current square's units map...
        for(var sui in cur_square_units){
            var cur_unit = cur_square_units[sui];
```

```
            for(var ui in cur_unit){
                var cur_unit_square = cur_unit[ui];

                if(cur_square_peers.indexOf(cur_unit_square) === -1 &&
                        cur_unit_square !== cur_square){
                    cur_square_peers.push(cur_unit_square);
                }
            }
        }

        // Save the current square an its associated peers to the map
        square_peers_map[cur_square] = cur_square_peers;
    }

    return square_peers_map;
};

sudoku._get_all_units = function(rows, cols){
    /* Return a list of all units (rows, cols, boxes)
    */
    var units = [];

    // Rows
    for(var ri in rows){
        units.push(sudoku._cross(rows[ri], cols));
    }

    // Columns
    for(var ci in cols){
       units.push(sudoku._cross(rows, cols[ci]));
    }

    // Boxes
    var row_squares = ["ABC", "DEF", "GHI"];
    var col_squares = ["123", "456", "789"];
    for(var rsi in row_squares){
        for(var csi in col_squares){
            units.push(sudoku._cross(row_squares[rsi], col_squares[csi]));
        }
    }

    return units;
};


// Conversions
//
----------------------------------------------------------------------
sudoku.board_string_to_grid = function(board_string){
    /* Convert a board string to a two-dimensional array
    */
    var rows = [];
    var cur_row = [];
    for(var i in board_string){
        cur_row.push(board_string[i]);
        if(i % 9 == 8){
```

```
            rows.push(cur_row);
            cur_row = [];
        }
    }
    return rows;
};

sudoku.board_grid_to_string = function(board_grid){
    /* Convert a board grid to a string
    */
    var board_string = "";
    for(var r = 0; r < 9; ++r){
        for(var c = 0; c < 9; ++c){
            board_string += board_grid[r][c];
        }
    }
    return board_string;
};


// Utility
//
// ----------------------------------------------------------------------

sudoku.print_board = function(board){
    /* Print a sudoku `board` to the console.
    */

    // Assure a valid board
    var report = sudoku.validate_board(board);
    if(report !== true){
        throw report;
    }

    var V_PADDING = " ";  // Insert after each square
    var H_PADDING = '\n'; // Insert after each row

    var V_BOX_PADDING = "  "; // Box vertical padding
    var H_BOX_PADDING = '\n'; // Box horizontal padding

    var display_string = "";

    for(var i in board){
        var square = board[i];

        // Add the square and some padding
        display_string += square + V_PADDING;

        // Vertical edge of a box, insert v. box padding
        if(i % 3 === 2){
            display_string += V_BOX_PADDING;
        }

        // End of a line, insert horiz. padding
        if(i % 9 === 8){
            display_string += H_PADDING;
        }
```

```
        // Horizontal edge of a box, insert h. box padding
        if(i % 27 === 26){
            display_string += H_BOX_PADDING;
        }
    }

    console.log(display_string);
};

sudoku.validate_board = function(board){
    /* Return if the given `board` is valid or not. If it's valid, return
    true. If it's not, return a string of the reason why it's not.
    */

    // Check for empty board
    if(!board){
        return "Empty board";
    }

    // Invalid board length
    if(board.length !== NR_SQUARES){
        return "Invalid board size. Board must be exactly " + NR_SQUARES +
                " squares.";
    }

    // Check for invalid characters
    for(var i in board){
        if(!sudoku._in(board[i], sudoku.DIGITS) && board[i] !==
sudoku.BLANK_CHAR){
            return "Invalid board character encountered at index " + i +
                    ": " + board[i];
        }
    }

    // Otherwise, we're good. Return true.
    return true;
};

sudoku._cross = function(a, b){
    /* Cross product of all elements in `a` and `b`, e.g.,
    sudoku._cross("abc", "123") ->
    ["a1", "a2", "a3", "b1", "b2", "b3", "c1", "c2", "c3"]
    */
    var result = [];
    for(var ai in a){
        for(var bi in b){
            result.push(a[ai] + b[bi]);
        }
    }
    return result;
};

sudoku._in = function(v, seq){
    /* Return if a value `v` is in sequence `seq`.
    */
    return seq.indexOf(v) !== -1;
```

```
};

sudoku._first_true = function(seq){
    /* Return the first element in `seq` that is true. If no element is
    true, return false.
    */
    for(var i in seq){
        if(seq[i]){
            return seq[i];
        }
    }
    return false;
};

sudoku._shuffle = function(seq){
    /* Return a shuffled version of `seq`
    */

    // Create an array of the same size as `seq` filled with false
    var shuffled = [];
    for(var i = 0; i < seq.length; ++i){
        shuffled.push(false);
    }

    for(var i in seq){
        var ti = sudoku._rand_range(seq.length);

        while(shuffled[ti]){
            ti = (ti + 1) > (seq.length - 1) ? 0 : (ti + 1);
        }

        shuffled[ti] = seq[i];
    }

    return shuffled;
};

sudoku._rand_range = function(max, min){
    /* Get a random integer in the range of `min` to `max` (non inclusive).
    If `min` not defined, default to 0. If `max` not defined, throw an
    error.
    */
    min = min || 0;
    if(max){
        return Math.floor(Math.random() * (max - min)) + min;
    } else {
        throw "Range undefined";
    }
};

sudoku._strip_dups = function(seq){
    /* Strip duplicate values from `seq`
    */
    var seq_set = [];
    var dup_map = {};
    for(var i in seq){
        var e = seq[i];
```

```
        if(!dup_map[e]){
            seq_set.push(e);
            dup_map[e] = true;
        }
    }
    return seq_set;
};

sudoku._force_range = function(nr, max, min){
    /* Force `nr` to be within the range from `min` to, but not including,
    `max`. `min` is optional, and will default to 0. If `nr` is undefined,
    treat it as zero.
    */
    min = min || 0
    nr = nr || 0
    if(nr < min){
        return min;
    }
    if(nr > max){
        return max;
    }
    return nr
}

// Initialize library after load
initialize();

// Pass whatever the root object is, lsike 'window' in browsers

let sudoku_generator = sudoku;

if (typeof module !== "undefined"){
      module.exports.sudoku_generator = sudoku_generator;
} else {
      eval("export {sudoku_generator};")

}
```