Sudoku Engine

Documentation

Thank you for purchasing my asset! For issue reports, feature suggestions, and other inquiries, please contact me via the methods listed below:

Unity Connect: Link

Email: cinationproject@gmail.com

Introduction

Sudoku Engine is a lightweight sudoku-generator script, designed to generate fully configurable 9x9 Sudoku puzzles, using custom prefabs.

The asset is designed to provide you with all the necessary tools to create your own Sudoku games/mini-games and can be used for both PC and Mobile projects.

The asset can be used without any prior coding experience, however, an intermediate level of understanding in Unity's C# is necessary to make modifications to the asset and implement additional features.

Functionality

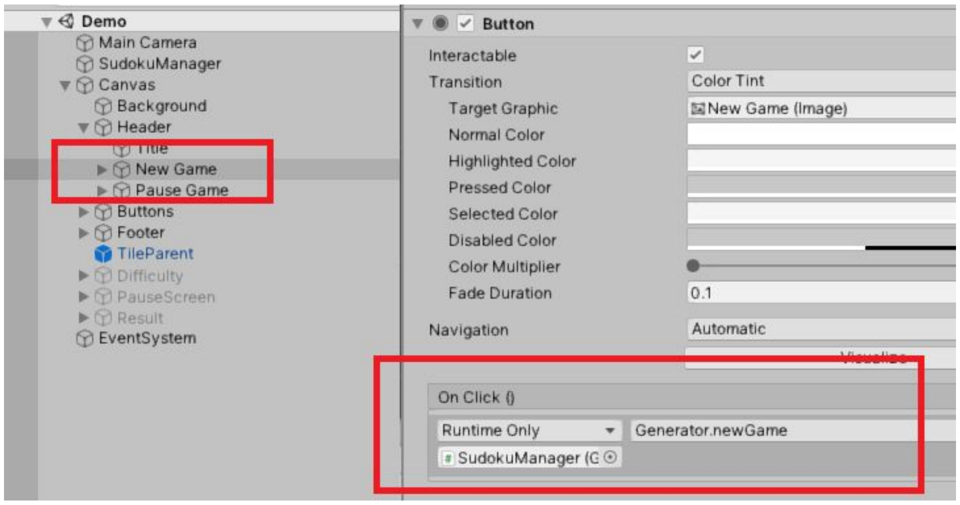
The asset has been designed to match the functionality of most sudoku games on the market. The UI is configured as follows:

1) You can generate a new puzzle by clicking on the **new game** button.
2) You can pause the game (and the timer in the bottom right corner) by clicking on the **pause** button.
3) The number buttons below the grid allow you to select the desired input.
4) The **notes** button switches to the **notes mode**, allowing you to take notes on any tiles.
5) Likewise, the erase button toggles the **erase mode,** allowing you to erase any answers or nodes (if the nodes mode is on).
6) The hints button will display a random hint when clicked.
7) You can view the number of mistakes made, select the difficulty, and see the number of remaining hints by accessing the bottom menu.

Quick Start

Sudoku Engine comes with a premade template scene. If you wish to skip the setup process, you can simply copy or modify the existing template to suit your game. The template scene can be found at **Scenes -> Template.**

The asset itself is designed to grant developers all the freedom they need to integrate it into their own projects. To access the asset's core functions, please navigate to **Scripts -> Generator.cs**, and carefully examine the script. Any **public** functions within the script can be freely called from other scripts (Generator.generator.**functionName()**), or be directly attached to buttons just like in the template (Fig. 1).

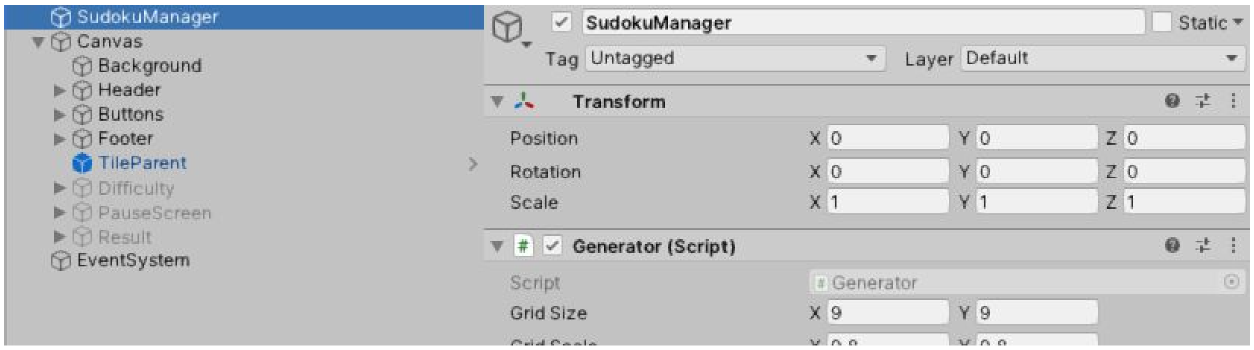


```
▼ Demo
    Main Camera
    SudokuManager
  ▼ Canvas
      Background
    ▼ Header
        Title
      ▶ New Game
      ▶ Pause Game
    ▶ Buttons
    ▶ Footer
      TileParent
    ▶ Difficulty
    ▶ PauseScreen
    ▶ Result
    EventSystem
```

```
▼ ✓ Button
Interactable            ✓
Transition              Color Tint
  Target Graphic        New Game (Image)
  Normal Color
  Highlighted Color
  Pressed Color
  Selected Color
  Disabled Color
  Color Multiplier
  Fade Duration         0.1
Navigation              Automatic
                        Visualize
On Click ()
Runtime Only ▼          Generator.newGame
SudokuManager (G ⊙
```

(Figure 1

If you wish to use the template as is, without editing or using the code, you can configure the template by opening the respective scene and clicking on the **SudokuManager** object (under canvas). The latter hosts the **Generator.cs** script, which contains all the asset's configurable parameters, while also containing references to necessary game objects.



```
  SudokuManager
▼ Canvas
    Background
  ▶ Header
  ▶ Buttons
  ▶ Footer
    TileParent
  ▶ Difficulty
  ▶ PauseScreen
  ▶ Result
  EventSystem
```

```
✓ SudokuManager                                   Static ▼
Tag Untagged ▼                     Layer Default ▼
▼ Transform
Position        X 0        Y 0        Z 0
Rotation        X 0        Y 0        Z 0
Scale           X 1        Y 1        Z 1
▼ # ✓ Generator (Script)
Script          Generator
Grid Size       X 9        Y 9
```

To use the asset on a brand-new scene, you can simply Drag & Drop the **Canvas** prefab from the **Prefabs** folder to your scene. The canvas comes with the Sudoku manager object, with a pre-configured **Generator** script. You may choose to merge your canvas with this prefab, or use the prefab itself as your canvas.

If you prefer using the script from scratch, simply attach the **Generator.cs** script from the **Scripts** folder to any game object on your scene. Afterwards, you will be able to configure the script from scratch, using your own UI elements and prefabs.

Generator Configuration

In order to properly use the asset, the generator must be properly configured. We shall go over each field of the script individually.

The script can be found in the **Scripts** folder and is by default attached to the **Sudoku Manager** object on the **Template** scene.

1) **Grid Size:** Is the size of the sudoku grid. Usual sudoku grids are 9x9. Please be careful when changing this value since the algorithm is not designed to generate grids larger than 9x9 (the process will take a very long time).

2) **Grid Scale:** If you wish to adjust the size of your grid on your X or Y axis, you can simply change the grid scale. A 1x1 scale means that the grid size will not change, while a 0.1x0.1 scale means that the grid will only be 10% of the original height and width.

3) **Neighborhood Size:** This is low large a neighborhood will be tile-wise. Usually sudoku grids have neighborhoods of size 3x3 for 9x9 grids. Usually the neighborhood size should be the grid size divided by 3. Again, it is not recommended to change the variable since some neighborhood sizes may make it impossible to generate solvable sudoku puzzles.

4) **Tile Prefab**: The prefab to be used as the tile template. You can create custom tiles as UI objects, with custom colors, backgrounds, etc.

5) **Background Prefab**: A prefab to be used as the background object. Please note that this object will be scaled based on the size of the generated grid and the tiles used.

6) **Neighborhood Background Prefab:** The prefab to be used as the neighborhood background object.

7) **Tile Parent:** This object will contain the generated UI elements (prefab instances).

8) **Fail Safe**: This option allows you to experiment with larger grids while avoiding crashes. However, setting the failsafe to a very high value may result in the engine freezing. By default, the failsafe should be set to around 10.000 iterations.

9) **Center Position**: The position where your grid should be centered. This position is by default relative to the parent object of your **Tile Parent.**

10) **Neighborhood Margin:** The distance between neighborhoods.

11) **Tile Margin:** The distance between tiles.

12) **Background Margin:** The extra padding on the background. Larger margin values will create a larger frame around the grid.

13) **Max Mistakes:** The maximum number of mistakes the player can make before the game ends.

14) **Max Hints:** The maximum number of hints the player can use.

15) **Difficulty Config:** An object which allows you to create custom difficulties. Each difficulty has two values, the **name** of the difficulty, and the probability that tiles (with answers) will be generated under said difficulty. The probabilities range from 0 to 1, and each neighborhood will have at least one tile generated by default.

    a. **Difficulty Menu:** Is the object which will be enabled when the user clicks on the difficulty selection button. By default, this object contains several pre-made buttons for each difficulty.

    b. **Difficulty Button Text**: By default, the button that is used to choose the difficulty also acts like a text field that displays said difficulty. This object is the text object attached to the said button.

16) **Button Config**: An object which contains various button configuration elements.

    a. **Main Button Default Color**: Main buttons are the **erase** and **notes** buttons. The main color is the color of these buttons in a non-selected state.

    b. **Main Button Selected Color:** The selected color is used to highlight the buttons when they are selected.

c. **Number Buttons Default Color:** The number buttons are the buttons 1 – 9, used to select the value to write. The default color is the color of all the number buttons except for the selected one.

d. **Number Buttons Selected Color:** The selected color is the color of the selected numbers button.

e. **Notes Button:** The button used to enable **notes** mode.

f. **Erase Button:** The button used to enable **erase** mode.

g. **Number Buttons:** A game object array which allows you to store all your number buttons.

17) **Tile Config:** The tile configuration allows you to configure the color of your tiles in different states.

    a. **Default Tile Color:** The default color of your tiles when you have not given an answer or are taking notes.

    b. **Selected Tile Color**: If **select mode** is enabled, this color will be applied to tiles before any answer is selected.

    c. **Wrong Tile Color:** The color of the tile which indicates that you have provided an incorrect answer.

    d. **Correct Tile Color:** The color that indicates that you have provided a correct answer.

    e. **Highlighted Tile Color:** The color all tiles in the same row, column, and neighborhood.

18) **Result Config:** Result screen configuration.

    a. **Victory Text:** The text to be displayed when the game has been won.

    b. **Loss Text:** The text to be displayed when the player has lost.

    c. **Result Object:** The game object to be enabled when the result is shown.

    d. **Result Text Object:** The text object containing the result message.

19) **Mistakes Object:** The text object which will display the number of mistakes made.

20) **Hints Object:** The text object which will display the number of hints used.

21) **Time Object:** The object which will display the timer.

22) **Pause Screen:** The object to be enabled when the player clicks on the **pause** button.

23) **Select Mode:** This mode allows selecting tiles before providing any answers. The selected tile will be simply highlighted with the color specified in **tileConfig.highlightedTileColor.**

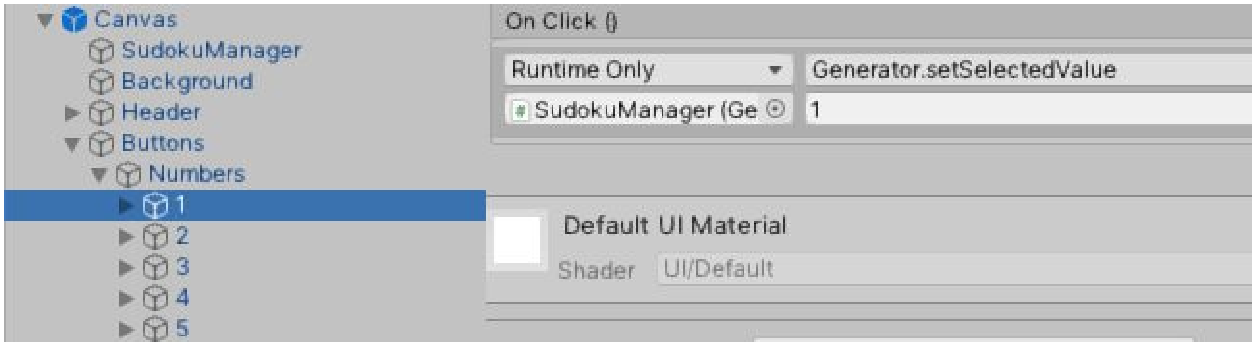
Object Configuration

If you choose to create the UI from scratch, please be aware of the following:

1) The pivot of the **Tile Parent** object must be set to 0,0.

2) Likewise, the pivot of the **Tile Prefab** must be also set to 0,0.

3) Most functions of the generator are directly attached to buttons (onClick events). Making the project from scratch implies that you will have to manually inspect the **Generator.cs** script and add the functions to the corresponding buttons. For instance, each number button under canvas has a function linked to it, with the corresponding integer passed as a parameter.



On Click ()

Runtime Only | Generator.setSelectedValue

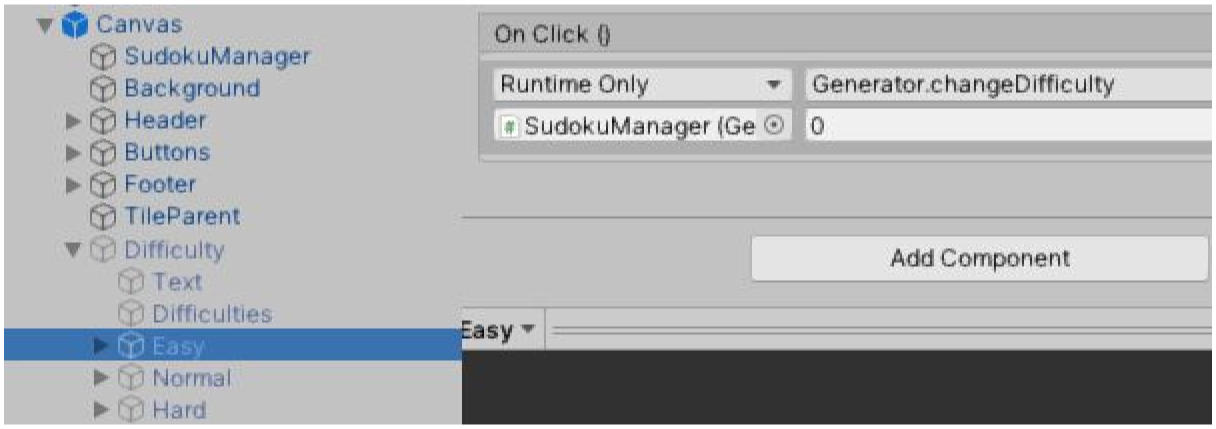
SudokuManager (Ge | 1

Default UI Material

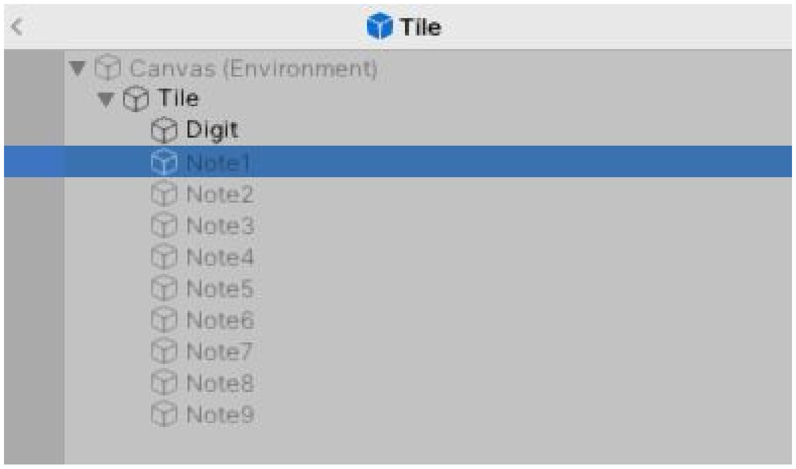
Shader UI/Default

4) You can add more difficulty settings by creating custom difficulty options. First, access your **Generator** script, and add your custom difficulties to the list (**DifficultyConfig)**. Afterwards, open your **Difficulty** game object and add new buttons corresponding to each difficulty. In the **OnClick** section of each button, add the object containing your **Generator** script to the object field and select the **changeDifficulty** function from the list of available functions. The parameter of each

function should correspond to the **index** of the associated difficulty option in the **DifficultyConfig.**



5) If you wish to change the number of note fields under individual tiles, you can do so by simply adding or deleting child text objects. By default, the number of your child objects should be equal to the size of your grid on the X axis (for a 9x9 grid you would need 9 child objects).



Conclusion

If you encounter any issues or find the documentation lacking, please contact me directly and I will address any issues as soon as possible.

Cination - Tsenkilidis Alexandros.

DOCUMENTATION END