

COLORED SWORDS DOCUMENTATION

HOW TO ADD NEW LEVEL

The levels in the game are made by separate scripts in the **/Scripts/LevelsCollection/** folder. Level instances are created and assigned in the **Game.cs** script in the **Start()** method.

The level script has two arrays **goalColors** and **newLevelMap**.

```
Color[,] goalColors = new Color[2, 3]
{
    { Colorz.Instance.Blue,  Colorz.Instance.Blue, Color.clear },
    { Colorz.Instance.Red,   Colorz.Instance.Red,  Color.clear },
};

Element[,] newLevelMap = new Element[2, 3]
{
    { new PaintableElement(),          new PaintableElement(),          new BluePaintElement(Vector2.one) },
    { new PaintableElement(),          new PaintableElement(),          new RedPaintElement(Vector2.one) },
};
```

The dimension of these arrays is the level size. **goalColors** contains the elements with which the field should be colored, this is the left side of the playing field. **newLevelMap** contains interactive elements (colored swords and fields for painting), this is the right side of the playing field.

The **goalColors** array is filled with the color elements contained in the **Colorz.cs** script:

```
public class Colorz : MonoBehaviourSingleton<Colorz>
{
    public Color Clear;
    public Color Blue;
    public Color Green;
    public Color Cyan;
    public Color Purple;
    public Color Red;
    public Color Yellow;
}
```

The **newLevelMap** array is filled with interactive elements (/Scripts/Elements/):

```
EmptyElement() // no interactive element
PaintableElement(int changeDirection) // empty cell to paint
BluePaintElement(Vector2 direction) // blue sword
GreenPaintElement(Vector2 direction) // green sword
LightBluePaintElement(Vector2 direction) // light blue sword
PurplePaintElement(Vector2 direction) // purple sword
RedPaintElement(Vector2 direction) // red sword
YellowPaintElement(Vector2 direction) // yellow sword
```

In the level class constructor, set the path to the level image:

```
LevelImage = "LevelImages/1"; - //Pictures must be in the /Resources/ folder
```

Once you've created the level script, open the **Game.cs** script.

In the **Start()** method, create a new instance of your level:

```
NewLevel NewLevel_1 = new NewLevel(); // Your class and field names may differ
```

Add an instance of your level to the collection of all levels:

```
AllLevels.Add(NewLevel_1);
```

Note that the Game object in the scene has a LastLevel field indicating the last level in the game.

VISUAL LEVEL EDITOR

You have the opportunity to visually build the level and play it (at the moment you can not save from the editor).

To enter the visual editor in the scene on the **Game** object in the **Game.cs** script, set the **DeveloperMode** field to true. Start the game, click on the first level. Below you will see a set of elements.



To start editing, check the "**Edit Mode**" box, which is located on the bottom panel on the left. Now you can select an interactive element and place it on the field. To rotate any element, hold down the "**R**" key and left-click on this element. To test the assembled level, uncheck "**Edit Mode**".

Please note that the visual editor was made for the developer and should not be available in the final game. Therefore, before building on the **Game** object in the **Game.cs** script, set the **DeveloperMode** field to false

HOW TO SKIP LEVEL

You can skip the running level, just press “F6” key (only works in the unity editor)