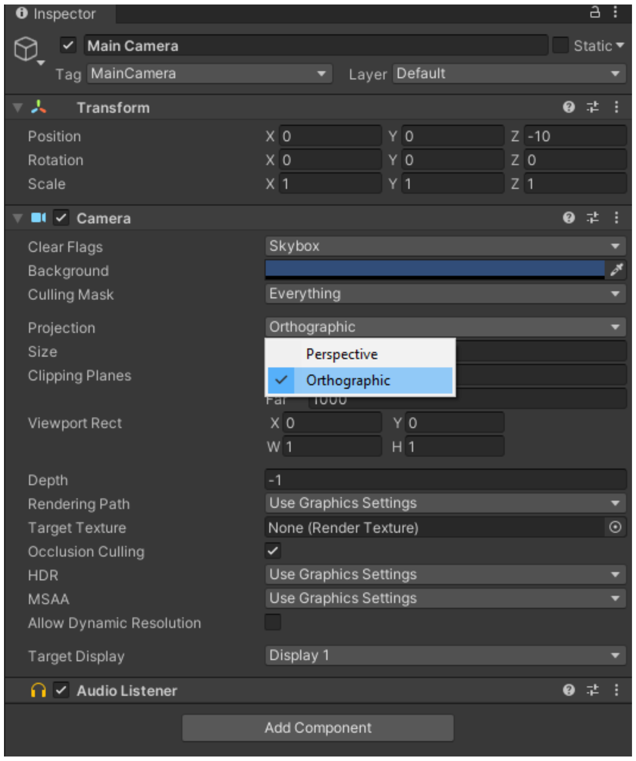
Table of Contents:

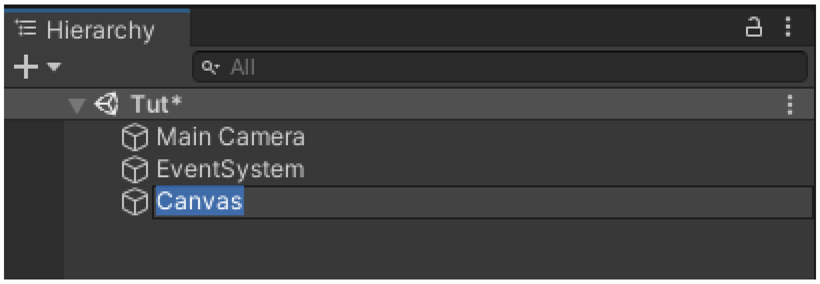
Setup:

1. Change Main Camera Orthographic.



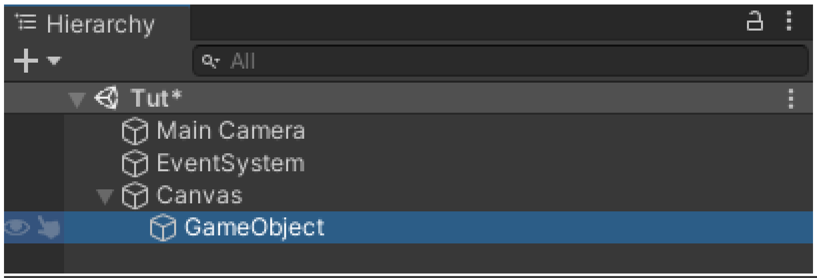
```
Inspector
Main Camera                                   Static
Tag MainCamera                 Layer Default
Transform
Position                X 0      Y 0      Z -10
Rotation                X 0      Y 0      Z 0
Scale                   X 1      Y 1      Z 1
Camera
Clear Flags             Skybox
Background
Culling Mask            Everything
Projection              Orthographic
Size                      Perspective
Clipping Planes         ✓ Orthographic
                        Far  1000
Viewport Rect           X 0      Y 0
                        W 1      H 1
Depth                   -1
Rendering Path          Use Graphics Settings
Target Texture          None (Render Texture)
Occlusion Culling       ✓
HDR                     Use Graphics Settings
MSAA                    Use Graphics Settings
Allow Dynamic Resolution
Target Display          Display 1
Audio Listener
Add Component
```

2. Create a Canvas in the Hierarchy.

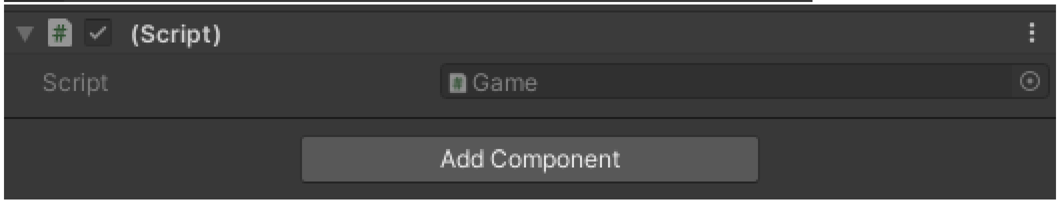


```
Hierarchy
All
Tut*
  Main Camera
  EventSystem
  Canvas
```

3. Create an GameObject to Contain the Grid and Add a Script Component with any Desired Name.

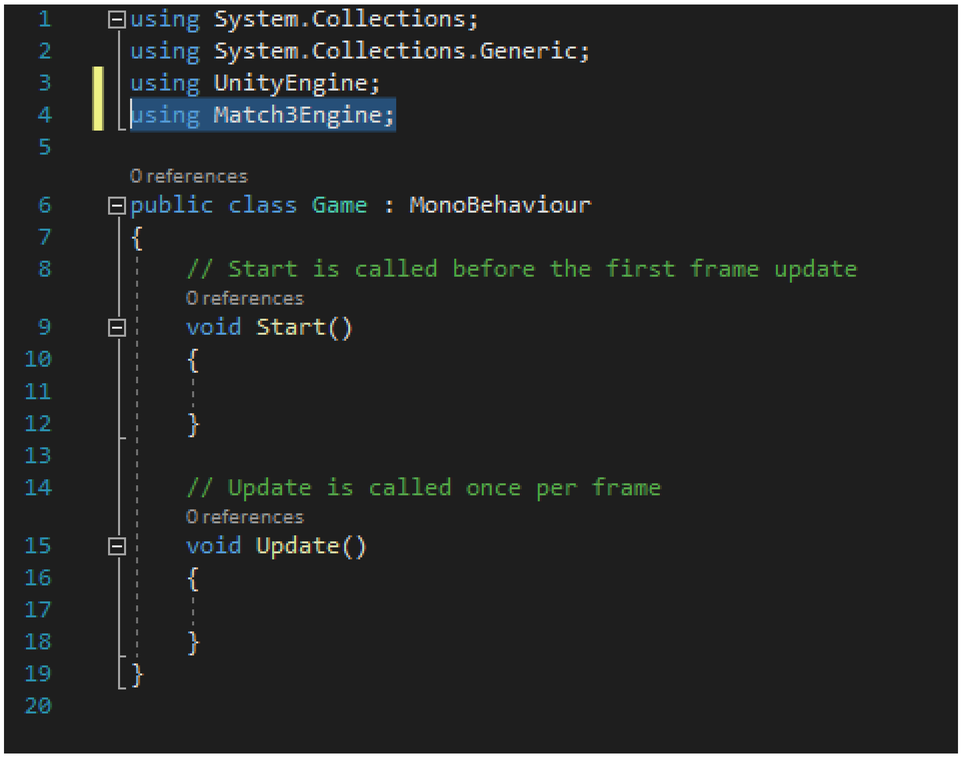


```
Hierarchy
All
Tut*
  Main Camera
  EventSystem
  Canvas
    GameObject
```



```
(Script)
Script                  Game
Add Component
```

4. Open the Script and Add the Match3Engine NameSpace.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Match3Engine;

0 references
public class Game : MonoBehaviour
{
    // Start is called before the first frame update
    0 references
    void Start()
    {

    }

    // Update is called once per frame
    0 references
    void Update()
    {

    }
}
```

Now you are ready to begin building your Match3 Game.

GridSettings

Basic Settings of the Grid.

GridSettings.Rows

Returns the Total Amount of Rows (Read Only) To Set Use [BuildGrid.Create].

GridSettings.Columns

Returns the Total Amount of Columns (Read Only) To Set Use [BuildGrid.Create].

GridSettings.GridContainer

Returns the Parent Object of Grid Elements(Read Only) To Set Use [BuildGrid.Create].

GridSettings.GridCanvas

Returns the Canvas Parent Object(Read Only) To Set Use [BuildGrid.Create].

GridSettings.SpaceSize

Returns size of the Spaces(Read Only) To Set Use [BuildGrid.Create].

GridSettings.SpaceLocationPos(int Row, int Column)

Returns the Local Position of Space(Read Only)[Parent: GridContainer] To Set Use [BuildGrid.Create].

GridSettings.SpaceBounds(int Row, int Column)

Returns the Local Position of Space TopLeft[0] and BottomRight[1](Read Only)[Parent: GridContainer].

GridSettings.MouseOverSpacePos()

Returns the Space Grid position Mouse is Over.

GridSettings.MousePressedSpacePos()

Returns the Space Grid position Mouse Pressed down on (Single Frame).

GridSettings.MouseReleasedSpacePos()

Returns the Space Grid position Mouse Released on (Single Frame).

GridSettings.MouseDirectionFromSpacePos(int Row, int Column)

Returns the Direction("Up","Down","Left","Right") Based on Space Position to Mouse Position Point Vector.

GridSettings.SpaceNextToPos(int Row, int Column, int MoveDown, int MoveUp, int MoveLeft, int MoveRight)

Returns a Space Position Given the Distance From Another Space Position.

How To Build A Grid:

To Create a Grid Use the Following Modules.

BuildGrid.Create(int Row, int Column, Vector2 SizeDelta, Transform Parent_Base)

Creates a Container for the Grid. Setups a Grid at the Parent Position.

- **Row:** Total Amount of Rows on the Grid.
- **Column:** Total Amount of Columns on the Grid.
- **SizeDelta:** Length and Width of Each Cell.
- **Parent_Base:** Parent Transform of the Grid; Parent Holds All of the Grid Elements.

Sample:

```
public GameObject GridParent;//Assign GameObject

void Build()
{
   BuildGrid.Create(10, 8, new Vector2(50,50), GridParent.transform);
}
void Start()
{
   Build();
}
```

BuildGrid.Elements[]

An Array of GameObjects to Appear on Grid; Array Numeral Position Represents Element Reference Number. Goes Up to 100 Elements.

Sample:

```
public GameObject GridParent;//Assign GameObject
public GameObject background1;//Assign Grid Element
public GameObject background2;//Assign Grid Element

void Build()
{
   BuildGrid.Create(10, 8, new Vector2(50,50), GridParent.transform);
   BuildGrid.Elements[0] = null; //An Empty Space
   BuildGrid.Elements[1] = background1;
   BuildGrid.Elements[2] = background2;

}
void Start()
{
   Build();
}
```

BuildGrid.PlaceElements(int LayerNum , int[,] GridElementPosition)

Creates a Layer and Places the GameObjects from **BuildGrid.Elements** on grid by the Array Numeral Position.

- **LayerNum**: The Layer to Place the GameObjects; 0 Being the Bottom.
- **GridElementPosition**: An Array matching the Rows and Columns of the Grid that contains the Array Numeral Position of GameObjects in **BuildGrid.Elements**.

Sample of **int[,] GridElementPosition**:

```
private int[,] GridBackground = new int[10, 8] //Must be Row and Columns of the Grid
    {
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1}

    };
```

```
BuildGrid.Elements[0] = null; //An Empty Space
BuildGrid.Elements[1] = background1;
BuildGrid.Elements[2] = background2;
```

Sample:

```
public GameObject GridParent;//Assign GameObject
public GameObject background1;//Assign Grid Element
public GameObject background2;//Assign Grid Element

private int[,] GridBackground = new int[10, 8]
    {
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1}

    };
void Build()
{
   BuildGrid.Create(10, 8, new Vector2(50,50), GridParent.transform);
   BuildGrid.Elements[0] = null; //An Empty Space
   BuildGrid.Elements[1] = background1;
   BuildGrid.Elements[2] = background2;

   BuildGrid.PlaceElements(1, GridBackground); //Emits the element during play

}
void Start()
{
   Build();
}
```

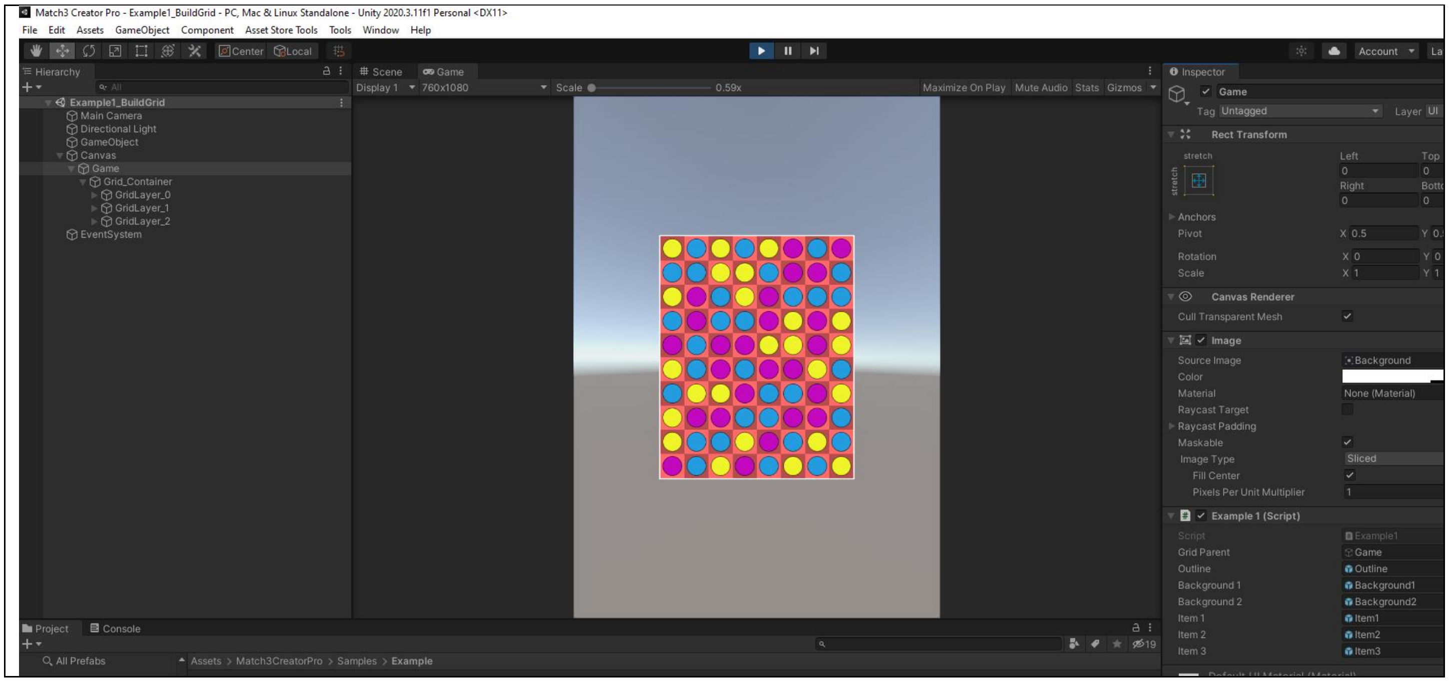
Creating Grid Sample (Example 1):

The Code Below Creates a Basic Grid. It Creates a Grid with Multiple Layers. **Example 1** can be found in the Samples Folder in the Match3Creator Asset.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Match3Engine;

public class Example1 : MonoBehaviour
{


    private int[,] GridBackground = new int[10, 8]
    {
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1}

    };





    private int[,] GridOutline = new int[10, 8]
    {
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3}

    };





    private int[,] GridItems = new int[10, 8]
    {
        {4, 5, 4, 5, 4, 6, 5, 6},
        {5, 5, 4, 4, 5, 6, 6, 5},
        {4, 6, 5, 4, 6, 5, 5, 5},
        {5, 6, 5, 5, 6, 4, 6, 4},
        {6, 5, 6, 6, 4, 4, 6, 4},
        {4, 5, 6, 5, 6, 6, 4, 5},
        {5, 4, 4, 6, 5, 5, 6, 4},
        {4, 6, 6, 5, 5, 6, 6, 5},
        {4, 5, 5, 4, 6, 5, 4, 5},
        {6, 5, 4, 6, 5, 4, 5, 4}

    };
```

```
public GameObject GridParent;

//Grid Elements
public GameObject outline;
public GameObject background1;
public GameObject background2;
public GameObject item1;
public GameObject item2;
public GameObject item3;

void Build()
{

    BuildGrid.Create(10, 8, background1.GetComponent<RectTransform>().sizeDelta, GridParent.transform);
    BuildGrid.Elements[0] = null;
    BuildGrid.Elements[1] = background1;
    BuildGrid.Elements[2] = background2;
    BuildGrid.Elements[3] = outline;
    BuildGrid.Elements[4] = item1;
    BuildGrid.Elements[5] = item2;
    BuildGrid.Elements[6] = item3;

    BuildGrid.PlaceElements(0, GridOutline);
    BuildGrid.PlaceElements(1, GridBackground);
    BuildGrid.PlaceElements(2, GridItems);

}

// Start is called before the first frame update
void Start()
{
    Build();
}
```

Output:



Match3 Creator Pro - Example1_BuildGrid - PC, Mac & Linux Standalone - Unity 2020.3.11f1 Personal <DX11>
File Edit Assets GameObject Component Asset Store Tools Tools Window Help
Center Local
Account
Hierarchy
Scene Game
Display 1 760x1080 Scale 0.59x Maximize On Play Mute Audio Stats Gizmos
Inspector
Project Console
All Prefabs
Assets > Match3CreatorPro > Samples > Example

```
Example1_BuildGrid
  Main Camera
  Directional Light
  GameObject
  Canvas
    Game
      Grid_Container
        GridLayer_0
        GridLayer_1
        GridLayer_2
  EventSystem
```

```
Game
Tag Untagged    Layer UI
Rect Transform
Left 0   Top 0
Right 0  Bottom 0
Pivot     X 0.5  Y 0.5
Rotation  X 0    Y 0
Scale     X 1    Y 1
Canvas Renderer
Cull Transparent Mesh  ✓
Image
Source Image     Background
Material         None (Material)
Raycast Target
Maskable         ✓
Image Type       Sliced
Fill Center      ✓
Pixels Per Unit Multiplier  1
Example 1 (Script)
Script        Example1
Grid Parent   Game
Outline       Outline
Background 1  Background1
Background 2  Background2
Item 1        Item1
Item 2        Item2
Item 3        Item3
```

Other Layer Settings:

BuildGrid.Layer[]

Get Information About a Layer.

BuildGrid.Layer[LayerNumber].AllElementsInLayer()
Returns all Elements in Layer.

BuildGrid.Layer[LayerNumber].AllElementsNumbersInLayer()
Returns all Element Numbers in Layer.

BuildGrid.Layer[LayerNumber].GetElementAtSpace(int Row, int Column)
Returns Element at Space in Layer.

BuildGrid.Layer[LayerNumber].GetElementNumAtSpace(int Row, int Column)
Get Element Number(Array Numeral Position) at Space in Layer.

BuildGrid.Layer[LayerNumber].SetElementNumAtSpace(int Row, int Column, int Number)
Set Element Number(Array Numeral Position) at Space in Layer. (Only Changes Number Reference).

BuildGrid.Layer[LayerNumber].SetClickableAtSpace(int Row, int Column , bool Set)
Set Whether Clickable at Space in Layer.

BuildGrid.Layer[LayerNumber].DeleteElementAtSpace(int Row, int Column, float Timer)
Delete Element at Space in Layer; Resets Space.

BuildGrid.Layer[LayerNumber].GetAllEmptySpaces()
Gets All Empty Spaces in Layer; Gets Spaces where Elements doesn’t Exist.

Partition the Grid:

This Feature Allows the User to Separate the Grid into Active and In-Active Parts.

BuildGrid.Partition.PartitionGrid(int Max_X, int Max_Y)

Separate Parts of a Grid. Must be Lower than Grid Rows and Columns. Doesn't Transition to a Part.

- **Max_X:** Amount of Rows to Partition.
- **Max_Y:** Amount of Columns to Partition.

Sample:

```
BuildGrid.Partition.PartitionGrid(4, 3); //Divides Grid into 4x3 Quadrant
```

BuildGrid.Partition.PartitionedParts

Returns the Total amount of Part Partitioned. Amount of Part depends on the Size of the Grid.

BuildGrid.Partition.TransitionPartPosition(int part, float Timer, bool InActiveUnclickable, bool InActiveTransparent)

Transition to Different Parts of the Grid. Centers the Part in the Parent Transform.

- **Part:** Part to Move Towards.
- **Timer:** Time to Move to Part.
- **InActiveUnclickable:** Make Spaces in Inactive Part Unclickable.
- **InActiveTransparent:** Make Spaces in Inactive Part Transparent.

Sample:

```
public int PartitionPart = 0; //Determines the part to move to. Changed by the user.
void TransitionPartition()
{
        if (PartitionPart < BuildGrid.Partition.PartitionedParts) // PartitionedParts is the Total Amount of Parts
         {
             BuildGrid.Partition.TransitionPartPosition(PartitionPart, .3f, true, true);
         }
}
```

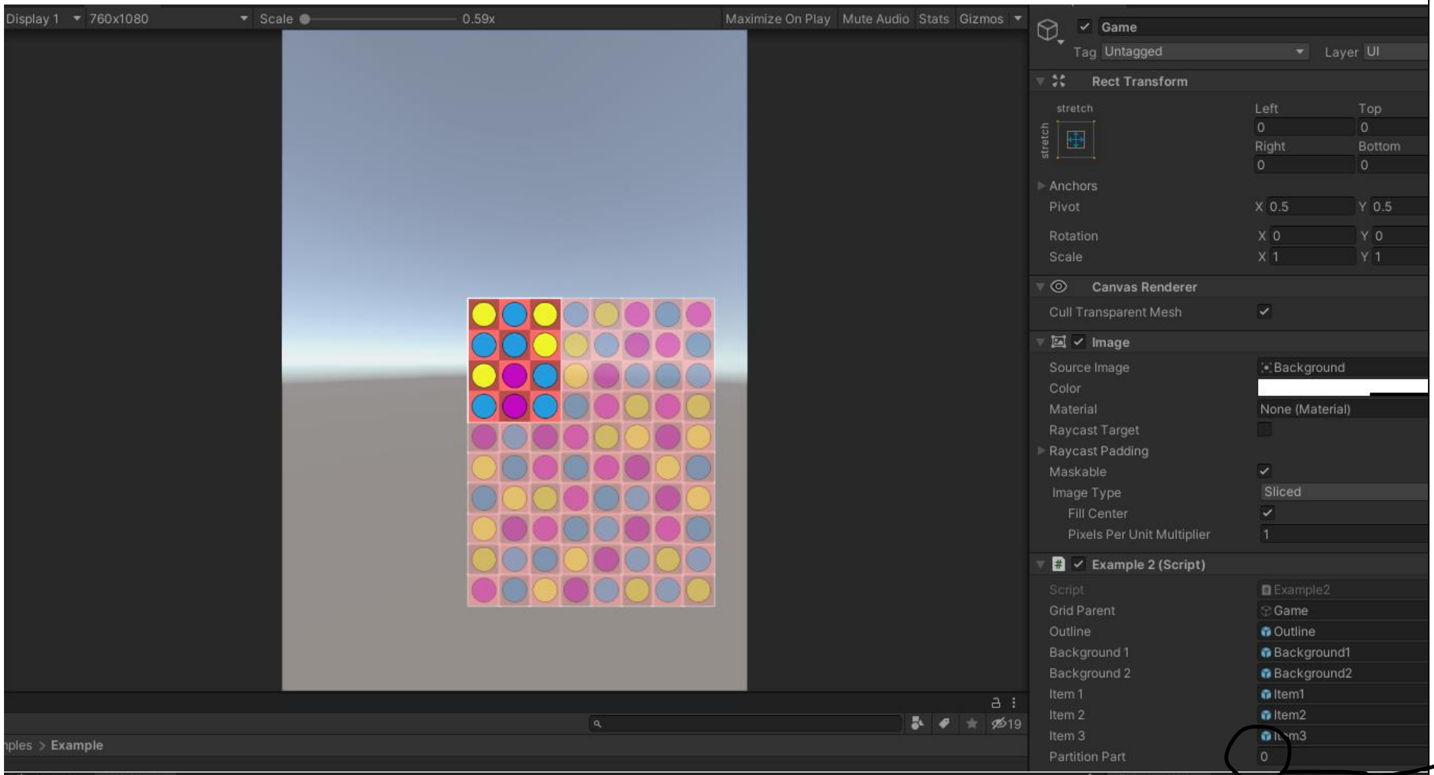
Creating A Partitioned Grid (Example 2):

The code below creates a partitioned grid that moves to different part depending on the number set by the user. **Example 2** can be found in the Samples Folder in the Match3Creator Asset.

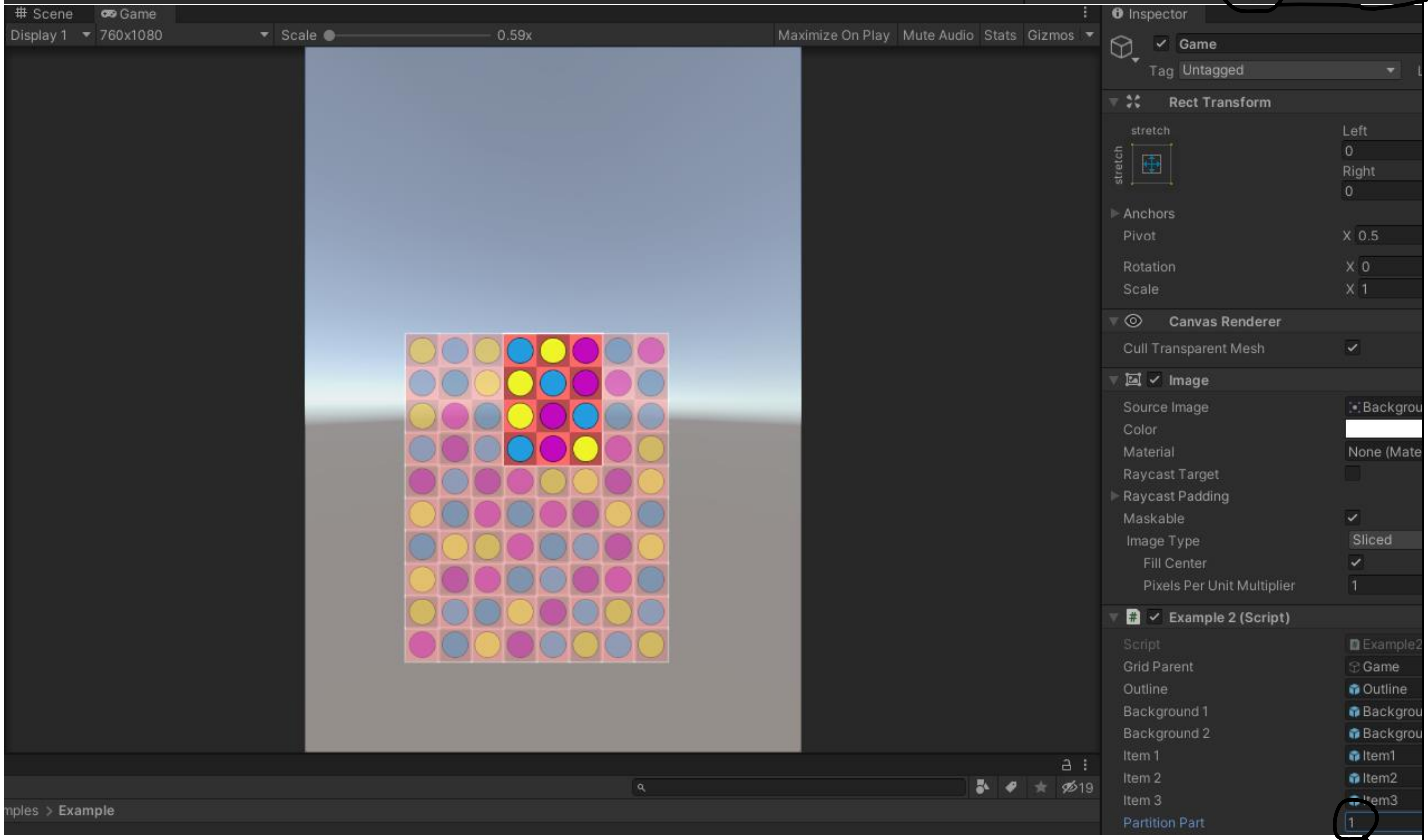
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Match3Engine;

public class Example2 : MonoBehaviour
{

    private int[,] GridBackground= new int[10, 8]
    {
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1}



    };

    private int[,] GridOutline = new int[10, 8]
    {
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3}



    };

    private int[,] GridItems = new int[10, 8]
{
        {4, 5, 4, 5, 4, 6, 5, 6},
        {5, 5, 4, 4, 5, 6, 6, 5},
        {4, 6, 5, 4, 6, 5, 5, 5},
        {5, 6, 5, 5, 6, 4, 6, 4},
        {6, 5, 6, 6, 4, 4, 6, 4},
        {4, 5, 6, 5, 6, 6, 4, 5},
        {5, 4, 4, 6, 5, 5, 6, 4},
        {4, 6, 6, 5, 5, 6, 6, 5},
        {4, 5, 5, 4, 6, 5, 4, 5},
        {6, 5, 4, 6, 5, 4, 5, 4}



};
```

```
    public GameObject GridParent;
    public GameObject outline;

    public GameObject background1;
    public GameObject background2;

    public GameObject item1;
    public GameObject item2;
    public GameObject item3;

    public int PartitionPart = 0;


    void Build()
    {

        BuildGrid.Create(10, 8, background1.GetComponent<RectTransform>().sizeDelta, GridParent.transform);
        BuildGrid.Elements[0] = null;
        BuildGrid.Elements[1] = background1;
        BuildGrid.Elements[2] = background2;
        BuildGrid.Elements[3] = outline;
        BuildGrid.Elements[4] = item1;
        BuildGrid.Elements[5] = item2;
        BuildGrid.Elements[6] = item3;

        BuildGrid.PlaceElements(0, GridOutline);
        BuildGrid.PlaceElements(1, GridBackground);
        BuildGrid.PlaceElements(2, GridItems);

        BuildGrid.Partition.PartitionGrid(4, 3);
    }



    void TransitionPartition()
    {
        if (PartitionPart < BuildGrid.Partition.PartitionedParts)
        {
            BuildGrid.Partition.TransitionPartPosition(PartitionPart, .3f, true, true);
        }

    }


    // Start is called before the first frame update
    void Start()
    {
        Build();
    }

    // Update is called once per frame
    void Update()
    {
        TransitionPartition();

    }
}
```

Output:



```
Game
Tag  Untagged                Layer  UI
Rect Transform
stretch                      Left   Top
                             0      0
                             Right  Bottom
                             0      0
Anchors
Pivot                        X 0.5  Y 0.5
Rotation                     X 0    Y 0
Scale                        X 1    Y 1
Canvas Renderer
Cull Transparent Mesh        ✓
Image
Source Image                 Background
Color
Material                     None (Material)
Raycast Target
Raycast Padding
Maskable                     ✓
Image Type                   Sliced
  Fill Center                ✓
  Pixels Per Unit Multiplier 1
Example 2 (Script)
Script                       Example2
Grid Parent                  Game
Outline                      Outline
Background 1                 Background1
Background 2                 Background2
Item 1                       Item1
Item 2                       Item2
Item 3                       Item3
Partition Part               0
```



```
Inspector
Game
Tag  Untagged
Rect Transform
stretch                      Left
                             0
                             Right
                             0
Anchors
Pivot                        X 0.5
Rotation                     X 0
Scale                        X 1
Canvas Renderer
Cull Transparent Mesh        ✓
Image
Source Image                 Backgrou
Color
Material                     None (Mate
Raycast Target
Raycast Padding
Maskable                     ✓
Image Type                   Sliced
  Fill Center                ✓
  Pixels Per Unit Multiplier 1
Example 2 (Script)
Script                       Example2
Grid Parent                  Game
Outline                      Outline
Background 1                 Backgrou
Background 2                 Backgrou
Item 1                       Item1
Item 2                       Item2
Item 3                       Item3
Partition Part               1
```

Number Changed to 1 from 0.

Other Partition Settings:

BuildGrid.Partition.DotweenCore
DoTween for Partition Movement.

BuildGrid.Partition.OnPart
Current Part Position.

BuildGrid.Partition.SetPartClickable(int part, bool on)
Enable/Disable all spaces in Part to be Clickable or Un-Clickable.

- **Part:** Part Partitioned.
- **On:** Enable or Disable if Clickable.

BuildGrid.Partition.Parts[(int)].positions
Returns Positions in Part.

BuildGrid.Partition.Parts[(int)].totalPositions
Return Total Amount of Positions in Part.

BuildGrid.Partition.Parts[(int)] .TopRightPosition
Returns The TopRight most Position.

BuildGrid.Partition.Parts[(int)] .BottomLeftPosition
The BottomLeft most Position.

Enable Swapping; Single Swap:

The User can Activate a Build-in Swap Module that enable to User to Move elements with a Swipe.

BuildSwap.SingleSwap.EngageSwap(int Layer, float Speed)

Enable the User to Swap Elements by Clicking and Moving the Mouse/Touch to the Next Position.

- **Layer**: Layer to Engage Swapping on.
- **Speed:** Duration of Swap.

Sample:

```
BuildSwap.SingleSwap.EngageSwap(2, .4f);
```

BuildSwap.Preset.LayerBoundary

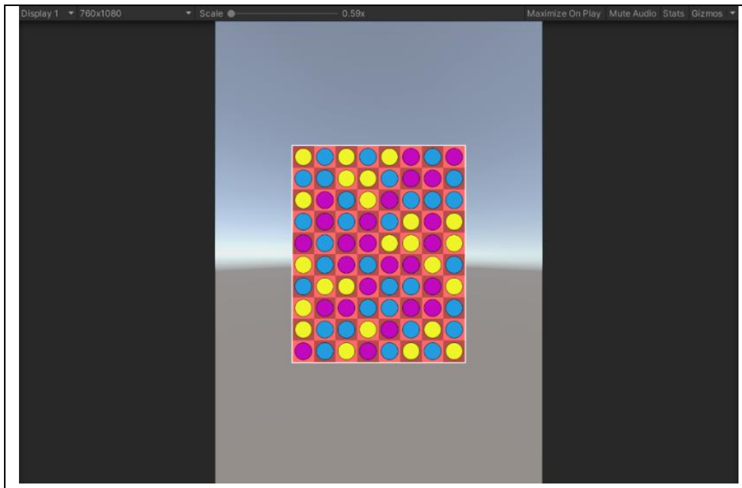
Swap on positions only where given layer elements exist. For example, if the user is swapping on layer 2 and the user set the LayerBoundary to 1, then the user can only swap layer 2 elements where layer 1 elements exist.

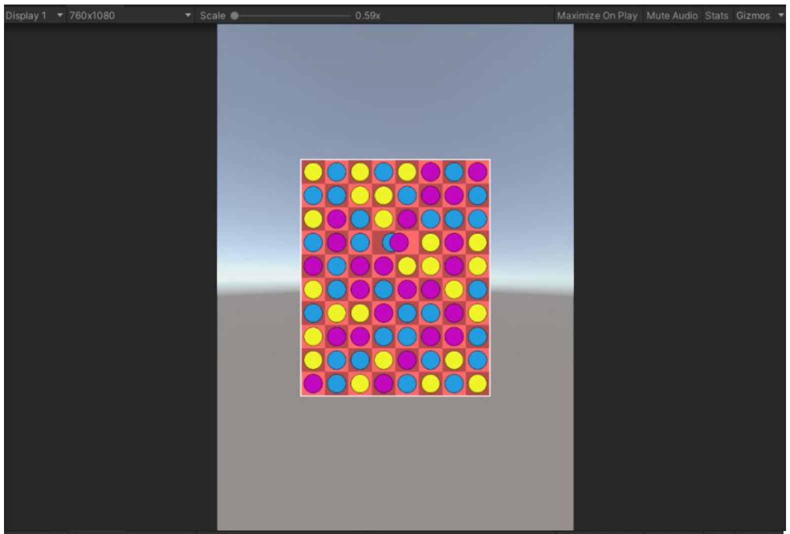
Enabling Single Swap (Example 3):

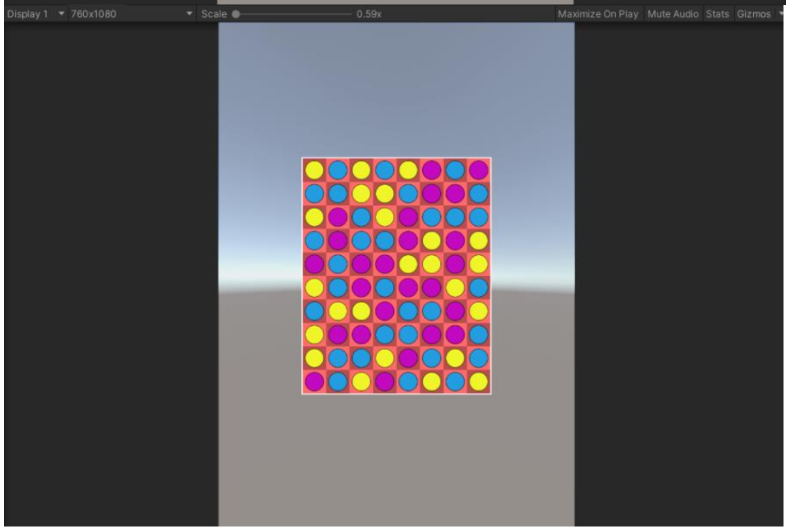
The code below enable single swapping. **Example 3** can be found in the Samples Folder in the Match3Creator Asset.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Match3Engine;

public class Example3 : MonoBehaviour
{
    private int[,] GridBackground = new int[10, 8]
{
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1}



};

    private int[,] GridOutline = new int[10, 8]
    {
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3}



    };

    private int[,] GridItems = new int[10, 8]
    {
        {4, 5, 4, 5, 4, 6, 5, 6},
        {5, 5, 4, 4, 5, 6, 6, 5},
        {4, 6, 5, 4, 6, 5, 5, 5},
        {5, 6, 5, 5, 6, 4, 6, 4},
        {6, 5, 6, 6, 4, 4, 6, 4},
        {4, 5, 6, 5, 6, 6, 4, 5},
        {5, 4, 4, 6, 5, 5, 6, 4},
        {4, 6, 6, 5, 5, 6, 6, 5},
        {4, 5, 5, 4, 6, 5, 4, 5},
        {6, 5, 4, 6, 5, 4, 5, 4}



    };
    public GameObject GridParent;
    public GameObject outline;

    public GameObject background1;
    public GameObject background2;

    public GameObject item1;
    public GameObject item2;
    public GameObject item3;


    void Build()
    {

        BuildGrid.Create(10, 8, background1.GetComponent<RectTransform>().sizeDelta, GridParent.transform);
        BuildGrid.Elements[0] = null;
        BuildGrid.Elements[1] = background1;
        BuildGrid.Elements[2] = background2;
        BuildGrid.Elements[3] = outline;
        BuildGrid.Elements[4] = item1;
        BuildGrid.Elements[5] = item2;
        BuildGrid.Elements[6] = item3;

        BuildGrid.PlaceElements(0, GridOutline);
        BuildGrid.PlaceElements(1, GridBackground);
        BuildGrid.PlaceElements(2, GridItems);

    }

    void SingleSwap()
    {

        //BuildSwap.SingleSwap.CanSwapWithEmptySpaces = false;
        BuildSwap.SingleSwap.EngageSwap(2, .4f);

    }

    // Start is called before the first frame update
    void Start()
    {
        BuildSwap.Preset.LayerBoundary = 1;
        Build();
    }

    // Update is called once per frame
    void Update()
    {

        SingleSwap();

    }

}
```

Output:


Display 1
760x1080
Scale
0.59x
Maximize On Play
Mute Audio
Stats
Gizmos


Display 1
760x1080
Scale
0.59x
Maximize On Play
Mute Audio
Stats
Gizmos


Display 1
760x1080
Scale
0.59x
Maximize On Play
Mute Audio
Stats
Gizmos

Other Single Swap Settings:

BuildSwap.SingleSwap.SingleEngaged
Returns a Whether Swap is Engaged.

BuildSwap.SingleSwap.PositionDragging
Swap Element While Mouse is Pressed.

BuildSwap.SingleSwap.CanSwapWithEmptySpaces
Can Swap with an Empty Space.

BuildSwap.SingleSwap.SingleSwapTween
DG Moving tween Extension(Only when Move is Engaged else is null)

Enable Swapping; Space Swap:

The User can Activate a Build-in Swap Module that enable to User to Move elements with a Swipe. With Space Swap, a click or tap on an element will move to an empty space next to the element.

BuildSwap.SpaceSwap.EngageSwap(int Layer, float Speed)

Enable the User to Swap Elements by clicking or tapping on a position.

- **Layer**: Layer to Engage Swapping on.
- **Speed:** Duration of Swap.

Sample:

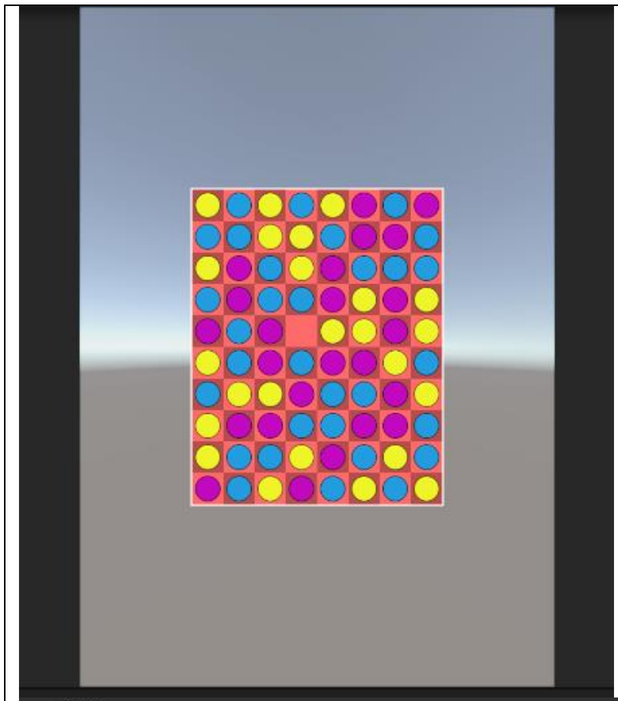
```
BuildSwap.SpaceSwap.EngageSwap(2, .4f);
```

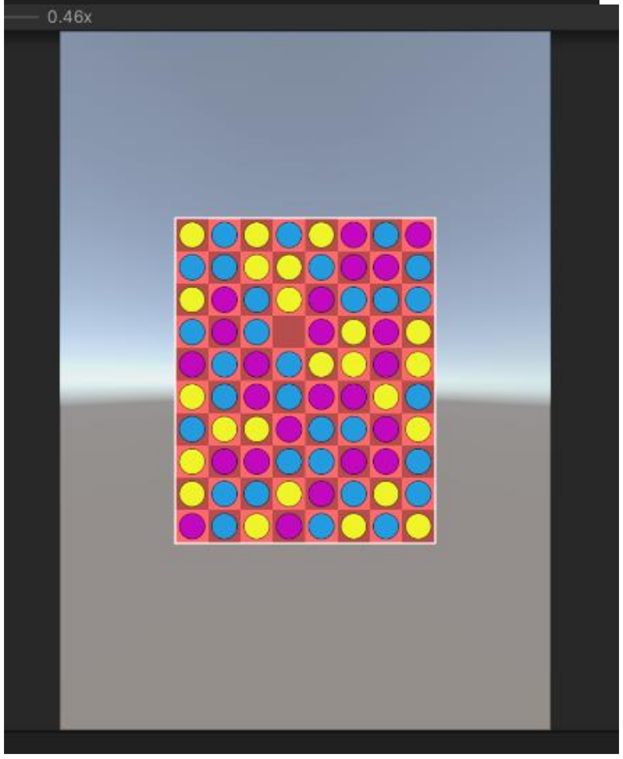
Enabling Space Swap (Example 4):

The code below enables space swapping. **Example 4** can be found in the Samples Folder in the Match3Creator Asset.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Match3Engine;

public class Example4 : MonoBehaviour
{
    private int[,] GridBackground = new int[10, 8]
{
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1}



};

    private int[,] GridOutline = new int[10, 8]
    {
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3}



    };

    private int[,] GridItems = new int[10, 8]
    {
        {4, 5, 4, 5, 4, 6, 5, 6},
        {5, 5, 4, 4, 5, 6, 6, 5},
        {4, 6, 5, 4, 6, 5, 5, 5},
        {5, 6, 5, 5, 6, 4, 6, 4},
        {6, 5, 6, 0, 4, 4, 6, 4},
        {4, 5, 6, 5, 6, 6, 4, 5},
        {5, 4, 4, 6, 5, 5, 6, 4},
        {4, 6, 6, 5, 5, 6, 6, 5},
        {4, 5, 5, 4, 6, 5, 4, 5},
        {6, 5, 4, 6, 5, 4, 5, 4}



    };
    public GameObject GridParent;
    public GameObject outline;

    public GameObject background1;
    public GameObject background2;

    public GameObject item1;
    public GameObject item2;
    public GameObject item3;


    void Build()
    {

        BuildGrid.Create(10, 8, background1.GetComponent<RectTransform>().sizeDelta, GridParent.transform);
        BuildGrid.Elements[0] = null;
        BuildGrid.Elements[1] = background1;
        BuildGrid.Elements[2] = background2;
        BuildGrid.Elements[3] = outline;
        BuildGrid.Elements[4] = item1;
        BuildGrid.Elements[5] = item2;
        BuildGrid.Elements[6] = item3;

        BuildGrid.PlaceElements(0, GridOutline);
        BuildGrid.PlaceElements(1, GridBackground);
        BuildGrid.PlaceElements(2, GridItems);

    }

    void SpaceSwap()
    {

        BuildSwap.SpaceSwap.EngageSwap(2, .4f);


    }

    // Start is called before the first frame update
    void Start()
    {
        Build();
    }

    // Update is called once per frame
    void Update()
    {

        SpaceSwap();

    }
}
```

Output:





Other Space Swap Setting:

BuildSwap.SpaceSwap.SpaceSwapEngaged
Returns a Whether Swap is Engaged.

BuildSwap.SpaceSwap.SpaceSwapTween
DG Moving tween Extension(Only when Move is Engaged else is null)

Other Swap Settings:

BuildSwap.Preset.PressedPosition_Vector2()
Returns the Pressed Position of the Grid.

BuildSwap.Preset.PressedPosition_Bool()
Returns True/False if Pressed on Position of the Grid.

BuildSwap.Preset.PositionIsAvailable(int Row, int Column)
Checks for Extra Ordinary Conditions that makes the Position Unable to be Interactable(Ex. Partition Grid).

BuildSwap.Preset.PositionSurroundings(int Row, int Column)
Returns an Array of Surrounding Positions (0 - Down , 1 - Up , 2 - Left , 3 - Right).

BuildSwap.Preset.GetPositionSwipeTowardsPosition(int Row, int Column, bool OutsideBounds)
Returns a Position Based of the Mouse Angle of the Given Position(Ex. If mouse position is Angled Left from the Given Position, it will return the position left the Given Position)

BuildSwap.Preset.LayerBoundary
Layer where the Swapping Occurs.

BuildSwap.Preset.SwapElements
Only Swap with given Elements.

BuildSwap.Preset.SwapElementsBoundary
Only Swap over given Elements in LayerBoundary.

BuildSwap.Preset.ElementNumbersPositionRestriction
Do Not Swap Elements that Share a position with these Element Numbers.

Setup Matching:

Using BuildMatch the User can Create there own Custom Matching and Check for matches.

BuildMatch.GetMatches(int Layer,out List<Vector2Int[]> Positions, out List<string> MatchNames, string Name = "default", string[] Names = null)

A Method that Finds and Outputs a List of Matches and their Names. The Default Match Pattern is 3 or more of the same element in a row or column, however this can be changed using SetCustomMatch().

- **Layer:** The Layer to Look for Matches.
- **Positions:** Output of a List of Grid Row and Column Positions that are in a Match Pattern.
- **MatchNames:** Output of a List of Names associated with the List of Positions.
- **Name:** Sets the Match Pattern to Look for; "default" is the built-in Match Pattern; Any other Match Pattern will need to be set by SetCustomMatch.
- **Names:** Sets Multiple types of Match Pattern to Look for; Will INGORE "Name" parameter if set.

Sample if Default Matching (The Default Match Pattern is 3 or more of the same element in a row or column):

```
List<Vector2Int[]> MatchPositions;
List<string> MatchNames;


BuildMatch.GetMatches(2, out MatchPositions, out MatchNames);
```

Sample if Single Custom Matching:

```
List<Vector2Int[]> MatchPositions;
List<string> MatchNames;

BuildMatch.GetMatches(2, out MatchPositions, out MatchNames, "Custom", null);
```

Sample if Multiple Custom Matching:

```
List<Vector2Int[]> MatchPositions;
List<string> MatchNames;

BuildMatch.GetMatches(2, out MatchPositions, out MatchNames, null, new string[] { "Custom",
"Custom2" });
```

BuildMatch.SetCustomMatch(string Name, Vector2Int[] Sequence, string[] Sequence_ElementNumber, int SequenceConfirmedAtArray)

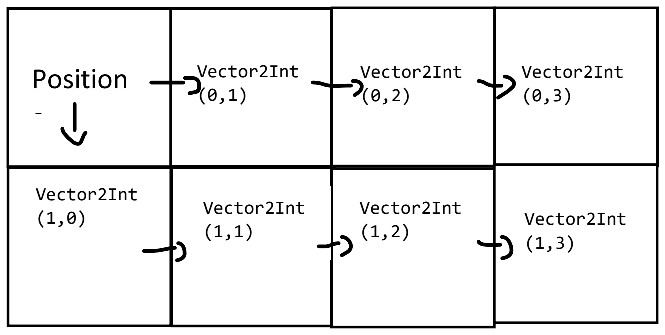
Add a Custom Match Pattern

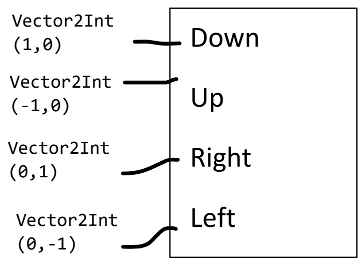
- **Name:** Set the Name of the Match Pattern. Will be used to call the custom match.
- **Sequence:** An Array of Positions to Set the Direction to Check for Matches.

```
Ex:

Vector2Int[] Sequence = new Vector2Int[]
            {
                    new Vector2Int(0,1),
                    new Vector2Int(0,2),
                    new Vector2Int(0,3),
                    new Vector2Int(1,0),
                    new Vector2Int(1,1),
                    new Vector2Int(1,2),
                    new Vector2Int(1,3)
            };
```

From any position on the grid, it will check the spaces for matches:
- 1 space right
- 2 space right
- 3 space right
- 1 space down
- 1 space down and 1 space right
- 1 space down and 2 space right
- 1 space down and 3 space right

Look at the graphic below for a visualization.



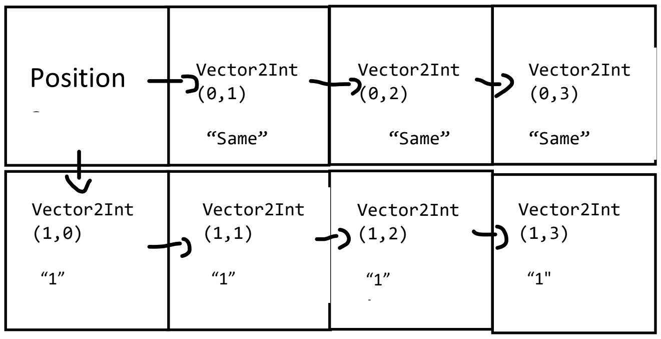


- **Sequence_ElementNumber:** This sets the Element expect to be found in the Sequence. Either put an element number or put "same" if you want it to be the same element number as the element at the position.

```
string[] Sequence_ElementNumber = new string[]
        {
                    "same",
                    "same",
                    "same",
                    "1",
                    "1",
                    "1",
                    "1",

        };
```

From the positions set from Sequence, it will check the spaces for Element Numbers (Array Numeral Position from BuildGrid.Elements[]). Must be same length as **Sequence**:

Look at the graphic below for visualization which follows **Sequence**:



- **SequenceConfirmedAtArray**: The Array Position which represents the Minimum Criteria to be a Complete Match Pattern.

Ex: If SequenceConfirmedAtArray = 1 (Array Position)

```
Vector2Int[] Sequence = new Vector2Int[]
            {
                    new Vector2Int(0,1),
                    new Vector2Int(0,2),
                    new Vector2Int(0,3),
                    new Vector2Int(1,0),
                    new Vector2Int(1,1),
                    new Vector2Int(1,2),
                    new Vector2Int(1,3)
            };
```

Confirmed match when positions confirmed

Will check but un-necessary for a confirmed match

Sample:

```
Vector2Int[] Sequence = new Vector2Int[]
{
                    new Vector2Int(0,1),
                    new Vector2Int(0,2),
                    new Vector2Int(0,3)
 };

string[] Sequence_ElementNumber = new string[]
{
                    "same",
                    "same",
                    "same"
 };

BuildMatch.SetCustomMatch("Custom", Sequence, Sequence_ElementNumber, 1);
```

Enabling Matching (Example 5):

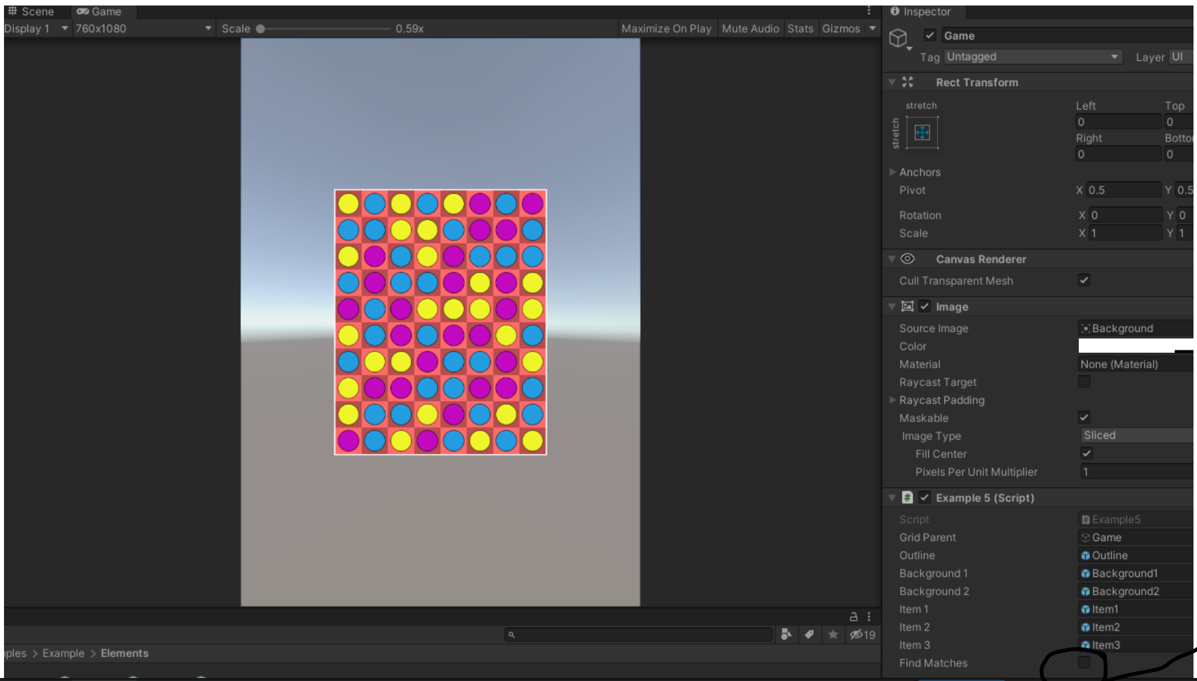
The code below sets up a custom match that checks for 3 in a row or 3 in a column for a match, performs a Dotween on the matches, and then final deletes it when user activate it. **Example 5** can be found in the Samples Folder in the Match3Creator Asset.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Match3Engine;
using DG.Tweening;

public class Example5 : MonoBehaviour
{
    private int[,] GridBackground = new int[10, 8]
{
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1}



};

    private int[,] GridOutline = new int[10, 8]
    {
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3}



    };

    private int[,] GridItems = new int[10, 8]
    {
        {4, 5, 4, 5, 4, 6, 5, 6},
        {5, 5, 4, 4, 5, 6, 6, 5},
        {4, 6, 5, 4, 6, 5, 5, 5},
        {5, 6, 5, 5, 6, 4, 6, 4},
        {6, 5, 6, 4, 4, 4, 6, 4},
        {4, 5, 6, 5, 6, 6, 4, 5},
        {5, 4, 4, 6, 5, 5, 6, 4},
        {4, 6, 6, 5, 5, 6, 6, 5},
        {4, 5, 5, 4, 6, 5, 4, 5},
        {6, 5, 4, 6, 5, 4, 5, 4}



    };
```

```
public GameObject GridParent;
public GameObject outline;

public GameObject background1;
public GameObject background2;

public GameObject item1;
public GameObject item2;
public GameObject item3;

void Build()
{

    BuildGrid.Create(10, 8, background1.GetComponent<RectTransform>().sizeDelta, GridParent.transform);
    BuildGrid.Elements[0] = null;
    BuildGrid.Elements[1] = background1;
    BuildGrid.Elements[2] = background2;
    BuildGrid.Elements[3] = outline;
    BuildGrid.Elements[4] = item1;
    BuildGrid.Elements[5] = item2;
    BuildGrid.Elements[6] = item3;

    BuildGrid.PlaceElements(0, GridOutline);
    BuildGrid.PlaceElements(1, GridBackground);
    BuildGrid.PlaceElements(2, GridItems);

}

void SingleSwap()
{
    //BuildSwap.SingleSwap.CanSwapWithEmptySpaces = false;
    BuildSwap.SingleSwap.EngageSwap(2, .4f);
}

void CustomMatch()
{
    Vector2Int[] Seq = new Vector2Int[]
        {
                new Vector2Int(0,1),
                new Vector2Int(0,2),
                new Vector2Int(0,3)
        };

    string[] Sequence_ElementNumber = new string[]
    {
                "same",
                "same",
                "same"
    };

    Vector2Int[] Seq2 = new Vector2Int[]
        {
                new Vector2Int(1,0),
                new Vector2Int(2,0),
                new Vector2Int(3,0)
        };

    string[] Sequence_ElementNumber2 = new string[]
    {
                "same",
                "same",
                "same"
    };

    BuildMatch.SetCustomMatch("Custom", Seq, Sequence_ElementNumber, 1);
    BuildMatch.SetCustomMatch("Custom2", Seq2, Sequence_ElementNumber2, 1);
}
```

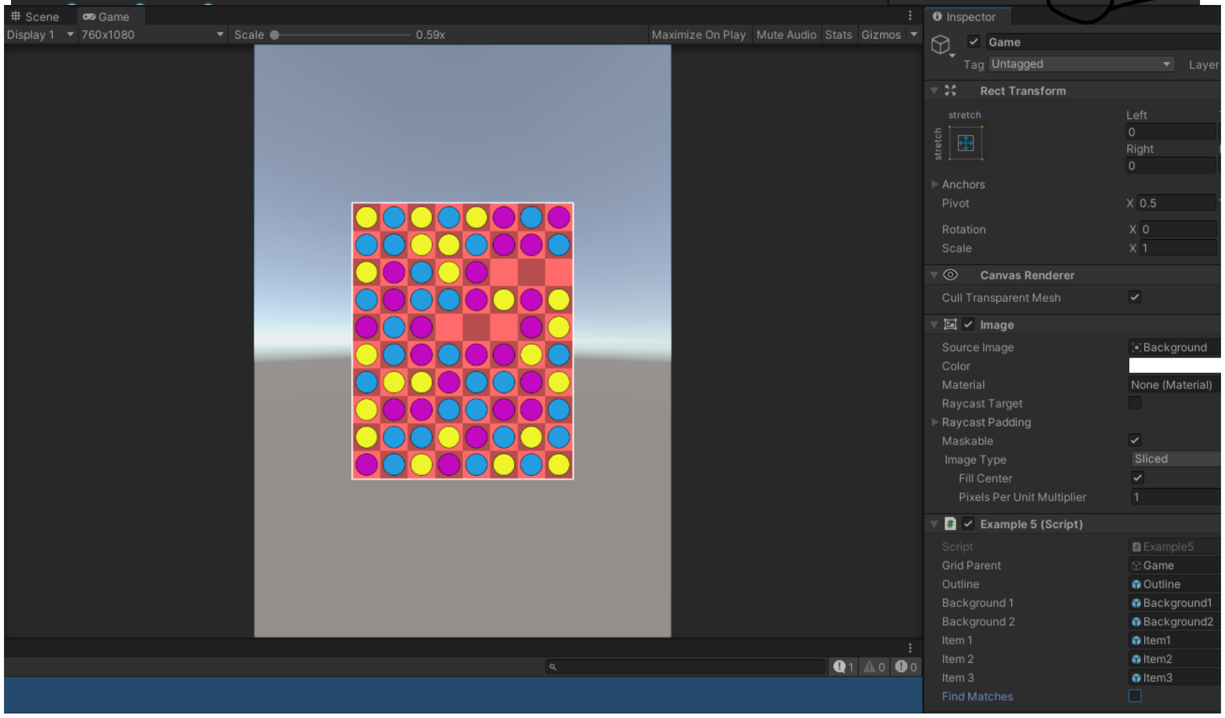
```
    public bool findMatches = false;

    public void GetMatches()
    {
        if (findMatches)
        {

            Debug.Log("Find Matches");

            List<Vector2Int[]> MatchPositions;
            List<string> MatchNames;

            BuildMatch.GetMatches(2, out MatchPositions, out MatchNames, null, new string[] { "Custom", "Custom2" });

            //Default
            //BuildMatch.GetMatches(2, out MatchPositions, out MatchNames);

            //List is not Empty
            if (MatchPositions != null)
            {
                //Get Match Positions
                foreach (Vector2Int[] array in MatchPositions)
                {
                    //Get Each Match position
                    foreach (Vector2Int pos in array)
                    {
                        var doing = BuildGrid.Layer[2].GetElementAtSpace(pos.x, pos.y).transform.DOShakeScale(.5f);
                        doing.OnComplete(() =>
                        {
                            BuildGrid.Layer[2].DeleteElementAtSpace(pos.x, pos.y, 0);
                        });

                    }

                }
            }



            findMatches = false;
        }



    }


    // Start is called before the first frame update
    void Start()
    {

        Build();
        CustomMatch();
    }

    // Update is called once per frame
    void Update()
    {
        GetMatches();
        SingleSwap();

    }
}
```

Output:



```
Game
Tag Untagged        Layer UI
Rect Transform
stretch             Left 0     Top 0
                    Right 0    Bottom 0
Anchors
Pivot               X 0.5   Y 0.5
Rotation            X 0     Y 0
Scale               X 1     Y 1
Canvas Renderer
Cull Transparent Mesh   ✓
Image
Source Image        Background
Color
Material            None (Material)
Raycast Target
Raycast Padding
Maskable            ✓
Image Type          Sliced
  Fill Center       ✓
  Pixels Per Unit Multiplier   1
Example 5 (Script)
Script              Example5
Grid Parent         Game
Outline             Outline
Background 1        Background1
Background 2        Background2
Item 1              Item1
Item 2              Item2
Item 3              Item3
Find Matches
```

Click Here



```
Game
Tag Untagged        Layer
Rect Transform
stretch             Left 0
                    Right 0
Anchors
Pivot               X 0.5
Rotation            X 0
Scale               X 1
Canvas Renderer
Cull Transparent Mesh   ✓
Image
Source Image        Background
Color
Material            None (Material)
Raycast Target
Raycast Padding
Maskable            ✓
Image Type          Sliced
  Fill Center       ✓
  Pixels Per Unit Multiplier   1
Example 5 (Script)
Script              Example5
Grid Parent         Game
Outline             Outline
Background 1        Background1
Background 2        Background2
Item 1              Item1
Item 2              Item2
Item 3              Item3
Find Matches
```

Fall and Emission:

The User can Customize Fall Paths and create positions where elements are emitted.

BuildMove.Fall.Engage(int Layer, float Speed)

Engage Fall of Elements on Layer to empty spaces. Default Fall Direction is Down.

- **Layer:** Layer to Make Elements Fall.
- **Speed:** Speed at Which Elements Fall.

Sample:

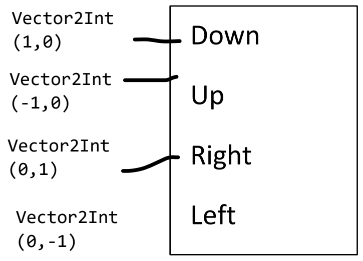
```
BuildMove.Fall.Engage(2, 5f);
```

BuildMove.Fall.FallingActive()

Returns if Falling is Active.

BuildMove.Fall.CustomFalldir[,].Directions

Create Custom Fall Path. The 2d Array is the Span total Rows and Columns. The Default Direction is Down. **Directions** is a new Vector2Int[] that can move Elements in multiple directions.



Sample:

```
//The 5 row and 0 Column; Will move Elements at position (5,0) Right.
BuildMove.Fall.CustomFalldir[5, 0].Directions = new Vector2Int[] { new Vector2Int(0, 1) };

//The 5 row and 0 Column; Will move Elements at position (5,0) Down or Right.
BuildMove.Fall.CustomFalldir[5, 0].Directions = new Vector2Int[] { new Vector2Int(1, 0), new Vector2Int(0, 1) };
```

BuildMove.Emit.Create(int Layer, int Row, int Column, int [] ElementNumbers)

- **Layer:** Layer to Emit Elements.
- **Row:** Row to Emit.
- **Column:** Column to Emit.
- **ElementNumbers:** Elements to Emit at Position Randomly. Number is (Array Numeral Position from BuildGrid.Elements[]).

Sample:

```
void EmitObj()
{
    for (int i = 0; i < 8; i++) //Top Row of the Grid
    {
        BuildMove.Emit.Create(2, 0, i, new int[] {  4, 5, 6 });
    }
}
```

Enable Fall and Emit (Example 6):

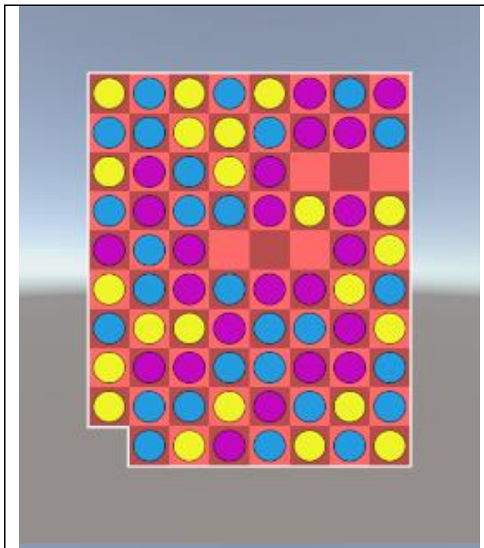
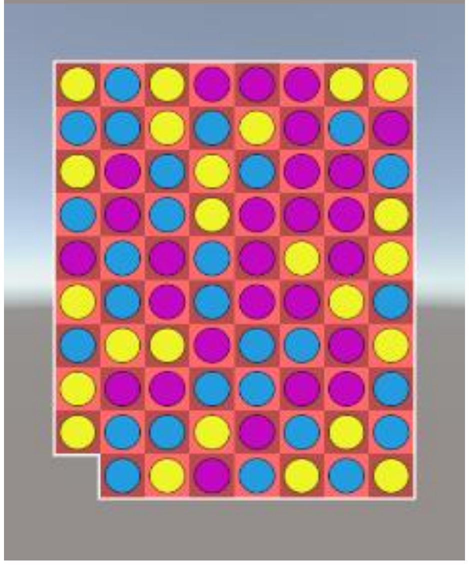
The code will emit Gameobjects at the top of the grid. The grid elements will move down except at the (5,0) position where it moves down and right due to custom.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Match3Engine;
using DG.Tweening;

public class Example6 : MonoBehaviour
{
    private int[,] GridBackground = new int[10, 8]
{
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {2, 1, 2, 1, 2, 1, 2, 1},
        {1, 2, 1, 2, 1, 2, 1, 2},
        {0, 1, 2, 1, 2, 1, 2, 1}
};

    private int[,] GridOutline = new int[10, 8]
    {
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {3, 3, 3, 3, 3, 3, 3, 3},
        {0, 3, 3, 3, 3, 3, 3, 3}

    };

    private int[,] GridItems = new int[10, 8]
    {
        {4, 5, 4, 5, 4, 6, 5, 6},
        {5, 5, 4, 4, 5, 6, 6, 5},
        {4, 6, 5, 4, 6, 5, 5, 5},
        {5, 6, 5, 5, 6, 4, 6, 4},
        {6, 5, 6, 4, 4, 4, 6, 4},
        {4, 5, 6, 5, 6, 6, 4, 5},
        {5, 4, 4, 6, 5, 5, 6, 4},
        {4, 6, 6, 5, 5, 6, 6, 5},
        {4, 5, 5, 4, 6, 5, 4, 5},
        {0, 5, 4, 6, 5, 4, 5, 4}

    };
```

```
public GameObject GridParent;
public GameObject outline;

public GameObject background1;
public GameObject background2;

public GameObject item1;
public GameObject item2;
public GameObject item3;


void Build()
{

    BuildGrid.Create(10, 8, background1.GetComponent<RectTransform>().sizeDelta, GridParent.transform);
    BuildGrid.Elements[0] = null;
    BuildGrid.Elements[1] = background1;
    BuildGrid.Elements[2] = background2;
    BuildGrid.Elements[3] = outline;
    BuildGrid.Elements[4] = item1;
    BuildGrid.Elements[5] = item2;
    BuildGrid.Elements[6] = item3;

    BuildGrid.PlaceElements(0, GridOutline);
    BuildGrid.PlaceElements(1, GridBackground);
    BuildGrid.PlaceElements(2, GridItems);

}

void SingleSwap()
{
    BuildSwap.SingleSwap.CanSwapWithEmptySpaces = true;
    BuildSwap.SingleSwap.EngageSwap(2, .4f);
}

void CustomMatch()
{
    Vector2Int[] Seq = new Vector2Int[]
        {
                new Vector2Int(0,1),
                new Vector2Int(0,2),
                new Vector2Int(0,3)
        };

    string[] Sequence_ElementNumber = new string[]
    {
                "same",
                "same",
                "same"
    };

    Vector2Int[] Seq2 = new Vector2Int[]
        {
                new Vector2Int(1,0),
                new Vector2Int(2,0),
                new Vector2Int(3,0)
        };

    string[] Sequence_ElementNumber2 = new string[]
    {
                "same",
                "same",
                "same"
    };

    BuildMatch.SetCustomMatch("Custom", Seq, Sequence_ElementNumber, 1);
    BuildMatch.SetCustomMatch("Custom2", Seq2, Sequence_ElementNumber2, 1);
}


public bool findMatches = false; //Activate
public void GetMatches()
{
    if (findMatches)
    {

        Debug.Log("Find Matches");

        List<Vector2Int[]> MatchPositions;
        List<string> MatchNames;

        BuildMatch.GetMatches(2, out MatchPositions, out MatchNames, null, new string[] { "Custom", "Custom2" });

        //BuildMatch.GetMatches(2, out MatchPositions, out MatchNames);


        foreach (Vector2Int[] array in MatchPositions)
        {
            foreach (Vector2Int pos in array)
            {
                var doing = BuildGrid.Layer[2].GetElementAtSpace(pos.x, pos.y).transform.DOShakeScale(.5f);
                doing.OnComplete(() => {
                    BuildGrid.Layer[2].DeleteElementAtSpace(pos.x, pos.y, 0);
                });

            }

        }



        findMatches = false;
    }



}

public void CustomFall()
{
    BuildMove.Fall.LayerBoundary = 1; //Will Fall Only Where Layer 1 Elements Are
    BuildMove.Fall.CustomFalldir[5, 0].Directions = new Vector2Int[] { new Vector2Int(1, 0), new Vector2Int(0, 1) };

}

void Falling()
{

    if (Fallf)
    {
        Debug.Log("Fall Along");
        Fallf = false;
        BuildMove.Fall.Engage(2, 5f);

    }

}

public bool Fallf = false; //Activate

void EmitObj()
{
    for (int i = 0; i < 8; i++) //Each Column
    {
        BuildMove.Emit.Create(2, 0, i, new int[] {  4, 5, 6 });
    }
}
```

```
    // Start is called before the first frame update
    void Start()
    {
        Build();
        CustomFall();
        CustomMatch();
        EmitObj();
    }

    // Update is called once per frame
    void Update()
    {
        Falling();
        GetMatches();
        SingleSwap();

    }
}
```

Output:

Other Fall Settings:

BuildMove.Fall.LayerBoundary
Fall on Positions Only where given Layer Elements Exist.

BuildMove.Fall.ExcludeFallElements
Exclude Elements from Falling.

BuildMove.Fall.ElementNumbersPositionRestriction
Do Not Fall Elements that Share a Position with these Element Numbers Regardless of Layer.

BuildMove.Fall.DotweenPaths
DoTween Path Core of all Falling Elements.

BuildMove.Fall.FallingActive()
Returns if Falling is Active.

BuildMove.SwitchElements
Switch the Elements Given Space; Moves the Element and switches Ref; Returns DoMove Tween for both elements.

BuildMove.Emit.CreateAtPosition(int Layer, int Row, int Column, int ElementNumber)
Emit a Single New Element at Position; Will erase element at position if exist.