DevLider

2D Puzzle Blocks

Documentation

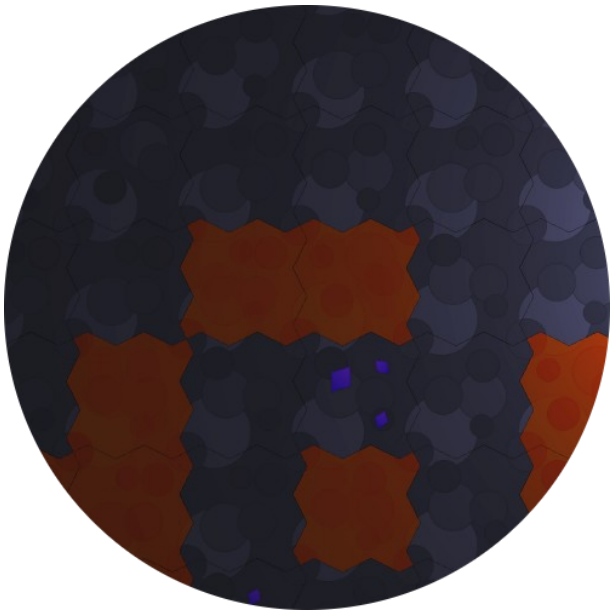


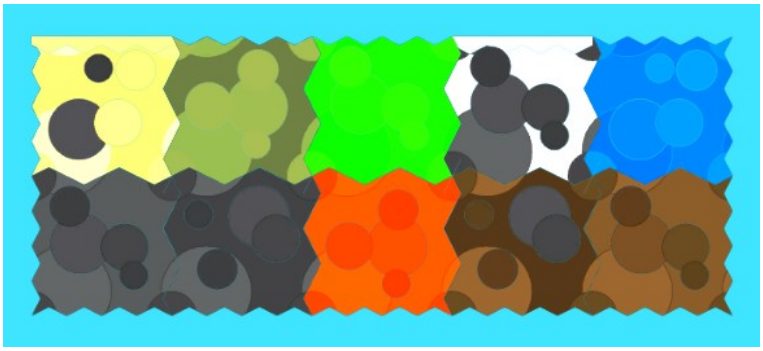
Table of Contents

1. Introduction

2D Puzzle Blocks is a package of simple puzzle textures for 2D games, with scripts for creating groups of blocks.

Package includes 10 textures:

- ground,
- gravel,
- lava,
- water,
- sand,
- toxin,
- stone,
- dark stone,
- ice stone,
- marsh.



Package contains textures in 32x32, 64x64, 128x128, 256x256, 512x512 PNG format, and SVG.

There are several scripts included in package, that are described in this document.

2. Getting Started

First of all, import asset to Your project. It is recommended to use Visual Studio IDE for C# developing. Scripts included in asset can be used for generating terrain for simple 2D games. It is required to create at least two scripts in Your project, for example:

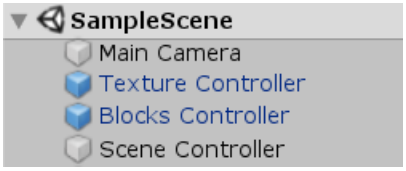
- **(Script)** Scene Controller – script which is using Blocks Controller as an object. It is equivalent to Game Controller in Unity Tutorials,
- **(Script)** Block – usually an empty class inheriting from Abstract Block. This script will be part of block prefab, which example can be found in Demo folder.

You also need to create some prefabs (can be copied from Demo/Prefabs):

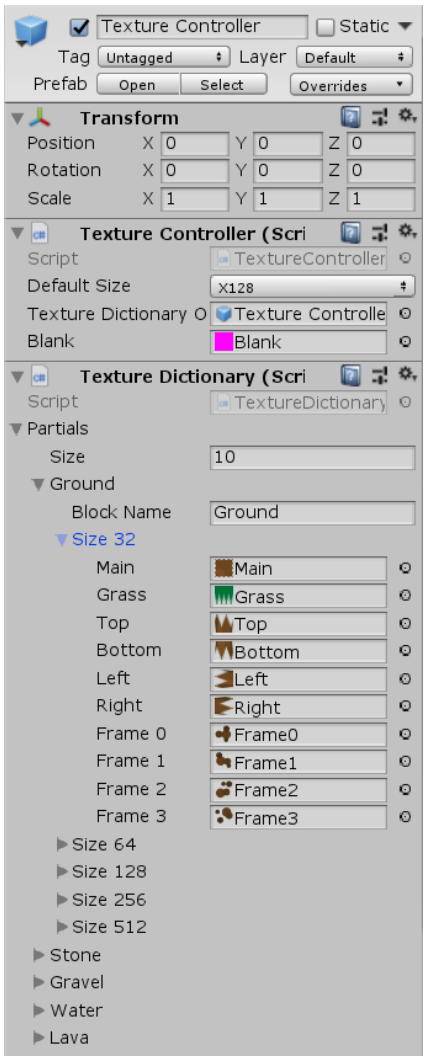
- **(Game Object)** Texture Controller
- **(Game Object)** Blocks Controller
- **(Game Object)** Block

It is highly recommended to copy prefabs from Demo folder.

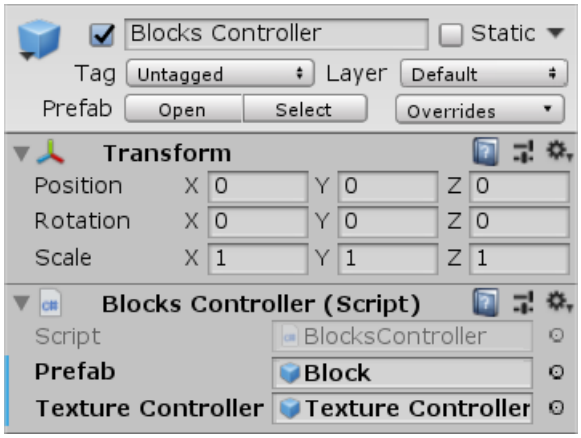
After creating essential objects, Your scene should look like on the screenshot below:



(Game Object) Texture Controller:



(Game Object) Blocks Controller:

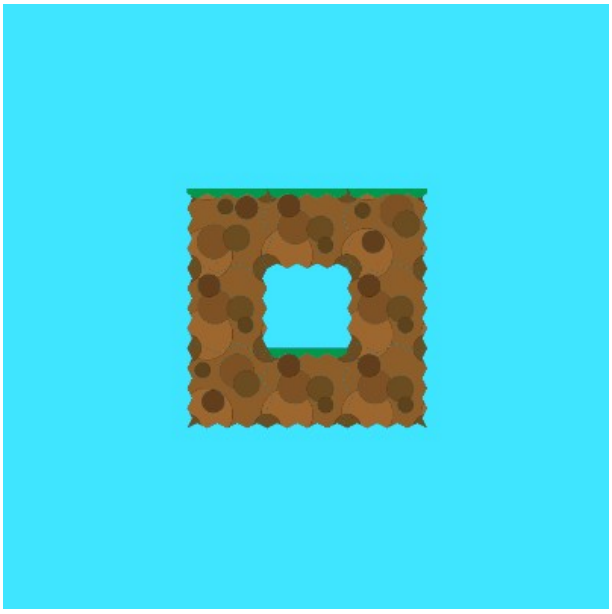


3. Example of usage

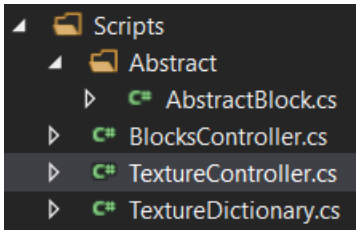
If you want to create a group of donut-shaped blocks, you need to define the block type, then prepare an array of types, and then call the *AddBlocks* method:

```
BlocksController blocksController = blocksControllerObject.GetComponent<BlocksController>();

//Set block size.
blocksController.SetSize(100f);

int groundBlock = blocksController.CreateBlock("Ground");

int[,] blocks = new int[3, 3];

blocks[0, 0] = groundBlock;
blocks[0, 1] = groundBlock;
blocks[0, 2] = groundBlock;
blocks[1, 0] = groundBlock;
blocks[1, 1] = 0;
blocks[1, 2] = groundBlock;
blocks[2, 0] = groundBlock;
blocks[2, 1] = groundBlock;
blocks[2, 2] = groundBlock;

blocksController.AddBlocks(blocks, Vector2.zero);
```

After running the code, You should see a ground donut:



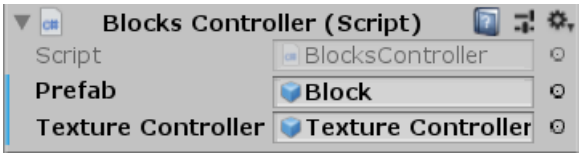
4. Asset script architecture



4.1. Controllers scripts

4.1.1. Blocks Controller

Blocks Controller script is used for managing blocks in scene. It is required to set up **(Game Object)** Block as a *prefab* and **(Game Object)** Texture Controller as a *textureController*.



Public methods:

void *SetSize:*

```
void SetSize(float size)
```

Setter for setting size of the blocks. Example:

```
blocksController.SetSize(100f);
```

int *CreateBlock:*

```
int CreateBlock(string name, bool liquid = false, bool animated = false)
```

This method is used to pre-define the blocks before they are created. Method is returning assigned ID of block type. Usage:

```
int groundBlock = blocksController.CreateBlock("Ground");
```

GameObject[,] *AddBlocks:*

```
GameObject[,] AddBlocks(int[,] blocks, Vector2 position)
```

Essential method used for creating chunk of blocks and returning them as an array of Game Objects. Example usage:

```
int[,] blocks = new int[3, 3];

blocks[0, 0] = groundBlock;
blocks[0, 1] = groundBlock;
blocks[0, 2] = groundBlock;
blocks[1, 0] = groundBlock;
blocks[1, 1] = 0;
blocks[1, 2] = groundBlock;
blocks[2, 0] = groundBlock;
blocks[2, 1] = groundBlock;
blocks[2, 2] = groundBlock;

blocksController.AddBlocks(blocks, Vector2.zero);
```

GameObject *AddBlock:*

```
GameObject AddBlock(int block, Vector2 position)
```

The same as an *AddBlocks*, but only creates a single block. Usage:

```
GameObject clone = AddBlock(2, Vector2.zero);
```

4.1.2. Texture Controller

Used for managing and caching textures. It creates List of ready to use Textures2D/Sprites. It is required to set up **(Game Object)** Texture Controller as a *textureDictionaryObject*, which using **(Script)** Texture Dictionary as a component.

Public methods:

void *Bake:*

```
void Bake(int id, string name)
```

User for baking/creating and caching texture from partials. Example usage:

```
textureController.Bake(2, "Stone");
```

void *Load:*

```
void Load(string key, TextureName name)
```

Loading cached texture and sprite to memory. Usage:

```
Load("Stone", TextureName.bottom);
```

Sprite *GetSprite:*

```
Sprite GetSprite(string key, TextureName name)
```

Returns the Sprite cached in memory by key and name. Usage:

```
Sprite sprite = GetSprite("Lava", TextureName.top);
```

Texture2D *GetTexture:*

```
Texture2D GetTexture(string key, TextureName name)
```

Returns the Texture2D cached in memory by key and name. Usage:

```
Texture2D texture = GetTexture("Lava", TextureName.top);
```

Sprite *GetBaked:*

```
Sprite GetBaked(int id, int frame, bool[] edges)
```

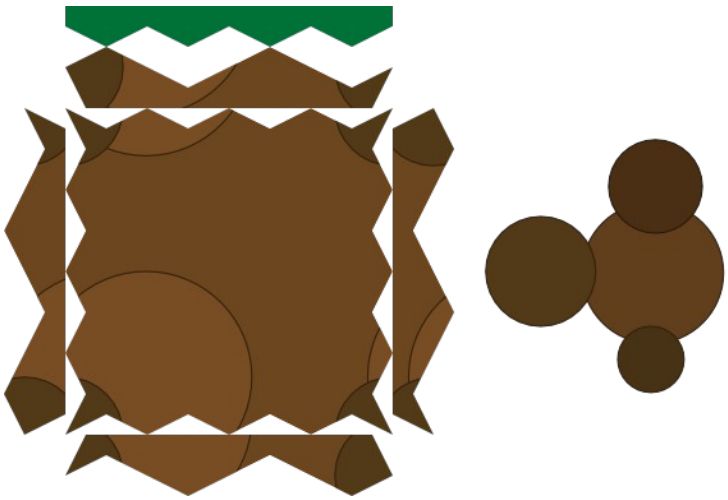
Returns the block Sprite previously cached in memory by *Bake* method. Usage:

```
sr.sprite = textureController.GetBaked(index, frame, edges);
```

4.2. Other

4.2.1. Texture Dictionary

Texture Dictionary is a dictionary that is storing all textures (partials) included in asset, used by Texture Controller script to generating sprites for blocks. The partials all shown in the picture below.



Texture Dictionary and other script may use Enum **TextureName** for naming partials:

```
public enum TextureName
{
	main,
	grass,
	top,
	bottom,
	left,
	right,

	frame0,
	frame1,
	frame2,
	frame3
}
```

Public methods:

Texture2D *GetPartial:*

```
public Texture2D GetPartial(string key, TextureName name, TextureSize size = TextureSize.x128)
```

Get Partial as Texture2D by **(string)** key, **(Enum.TextureName)** name and **(Enum.TextureSize)** size. Example usage:

```
Texture2D texture = textureDictionary.GetPartial("Ground", TextureName.main);
```

BlockPartials *GetPartials:*

```
public BlockPartials GetPartials(string key, TextureSize size = TextureSize.x128)
```

Get group of partials as struct PartialsGroup by **(string)** key and **(Enum.TextureSize)** size. Example usage:

```
BlockPartials partials = textureDictionary.GetPartials("Stone");
```

TextureName *GetTextureFrameName:*

```
public TextureName GetTextureFrameName(int i)
```

Get Enum.TextureName by frame id (0-3). Example usage:

```
TextureName name = textureDictionary.GetTextureFrameName(2);
```

4.2.2. Abstract Block

This is an abstraction on the basis of which you should create your own script for handling blocks. An inheritance class can be basically empty.

Public methods:

void *SetTextureController:*

```
public void SetTextureController(TextureController textureController)
```

Important method used as a dependency injection. Usage:

```
abstractBlock.SetTextureController(textureController);
```

***void** Set:*

```
void Set(int id, Color color)
```

Should be used after creating block and setting Texture Controller dependency injection. Method is setting id and color (optional) for block. Example usage:

```
abstractBlock.Set(1);
```

***void** RefreshTexture:*

```
void RefreshTexture()
```

Used for refreshing block texture. Example usage:

```
abstractBlock.RefreshTexture();
```

***void** SetLiquid:*

```
void SetLiquid(bool liquid = true)
```

Setter for changing liquid state of the block.

***void** SetAnimated:*

```
void SetAnimated(bool animated = true)
```

Setter for changing animation state of the block.

***void** SetNeighbor:*

```
void SetNeighbor(int indexX, int indexY, bool state = true)
```

This method is used to define the neighbors of the block and their coordinates. Example usage, which means that block has got a neighbor on the left side:

```
abstractBlock.SetNeighbor(-1, 0);
```

5. Contact

If You have any questions, please contact me:

Representative person:	Szymon Nowak
Email:	szynow@protonmail.com
Phone:	+48 782 390 575
Website:	https://devlider.com