Candy Fairy Tales

Hello! My name is **Viktor Yurov.** I'm the developer of this **Match-Three** template. So if you have any questions, feel free to ask. But first, I want to ask you two things.

- If you found a bug, check this page. There is a chance that the issue is known and maybe even have a fix.
- If you already bought the asset and contact me first time, please, write me ***invoice number*** of your purchase. It is necessary for validation you as a real customer.

Also I recommend you to read the online version of this guide — it is more actual and has useful GIF-animation instead of static pictures, that presented here.

Here is my contacts:

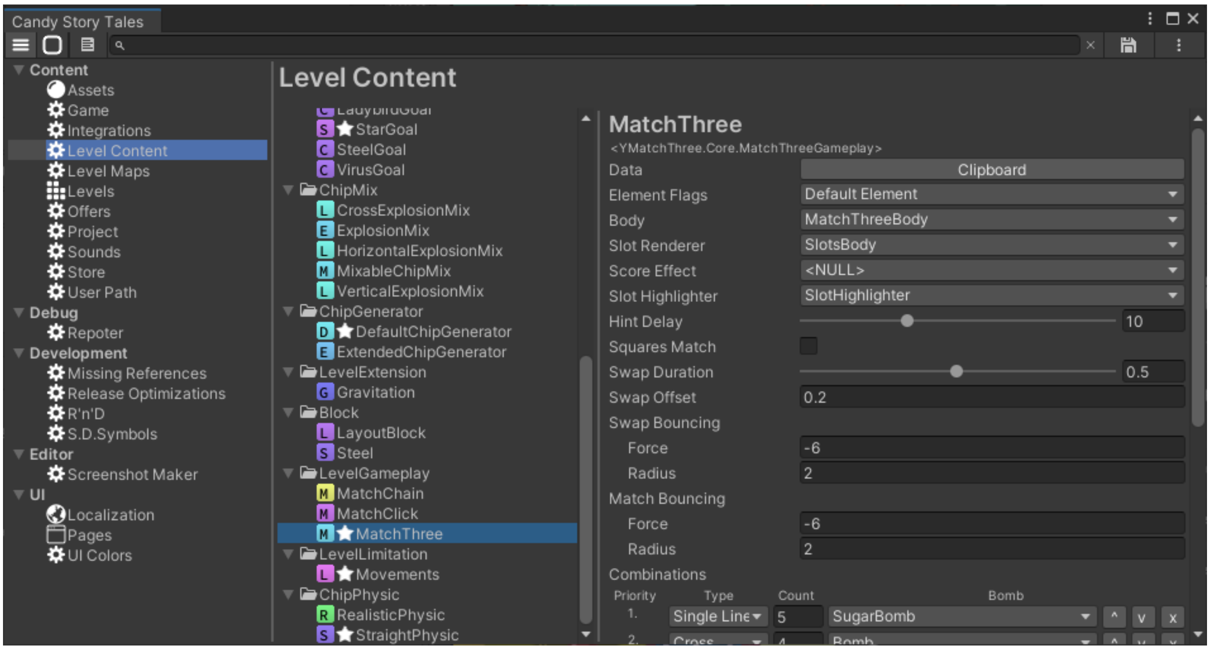
💌yurowm@gmail.com

💬https://t.me/yurowm

🙂https://vk.com/yurowm

Dashboard

The **Dashboard** is a main editor of the **Candy Fairy Tales** project. It contains all other editors, which allow to edit the game content. To open the **Dashboard** go to menu bar: *Yurowm > Dashboard*. The editor has the same title as the project name. Also there is the game icon that is used as the logo. The list of all the editors can be found in the right part of the **Dashboard**.

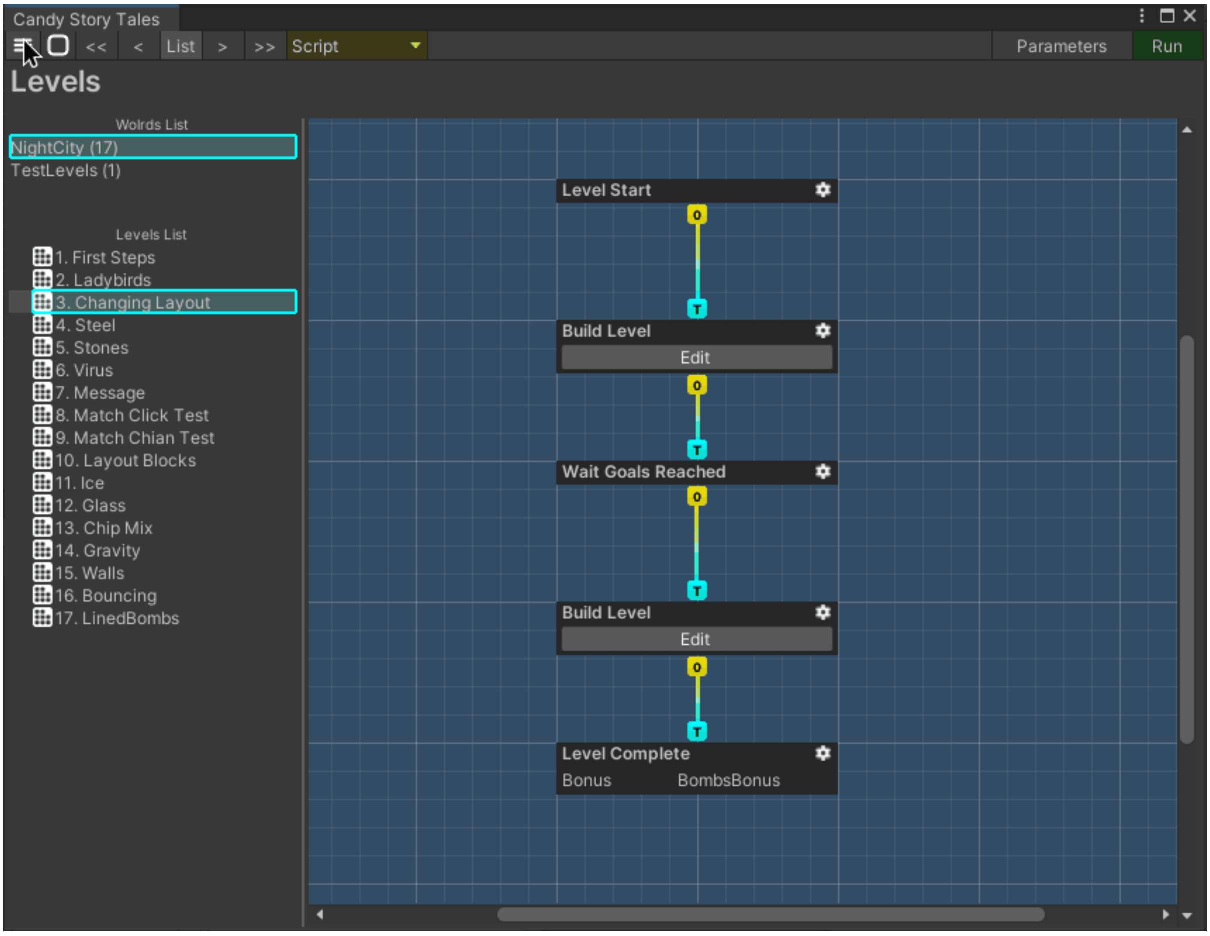


MatchThree
<YMatchThree.Core.MatchThreeGameplay>

Data: Clipboard
Element Flags: Default Element
Body: MatchThreeBody
Slot Renderer: SlotsBody
Score Effect: <NULL>
Slot Highlighter: SlotHighlighter
Hint Delay: 10
Squares Match: □
Swap Duration: 0.5
Swap Offset: 0.2
Swap Bouncing
Force: -6
Radius: 2
Match Bouncing
Force: -6
Radius: 2
Combinations

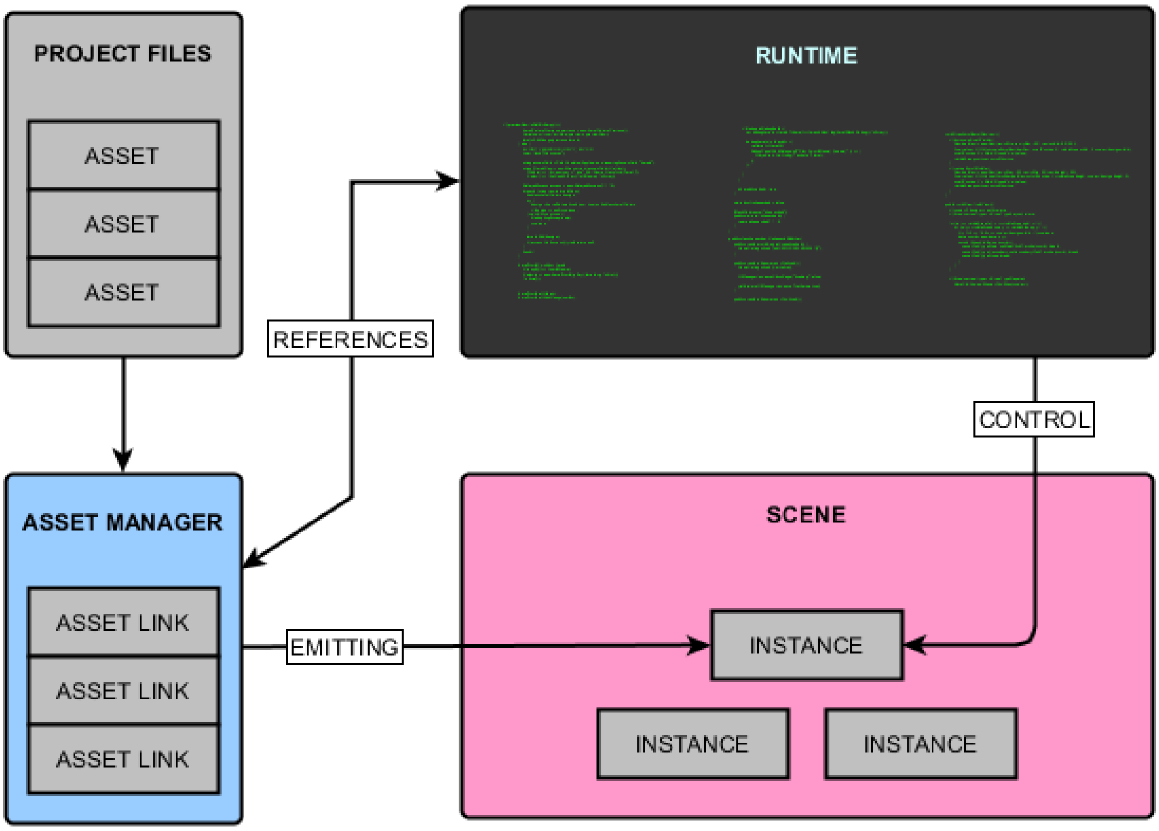
Priority	Type	Count	Bomb
1.	Single Line	5	SugarBomb
2.	Cross	4	Bomb

There is also two useful buttons in the top left corner, which helps to hide the list of the editor and make the **Dashboard** window maximized.



Asset Manager

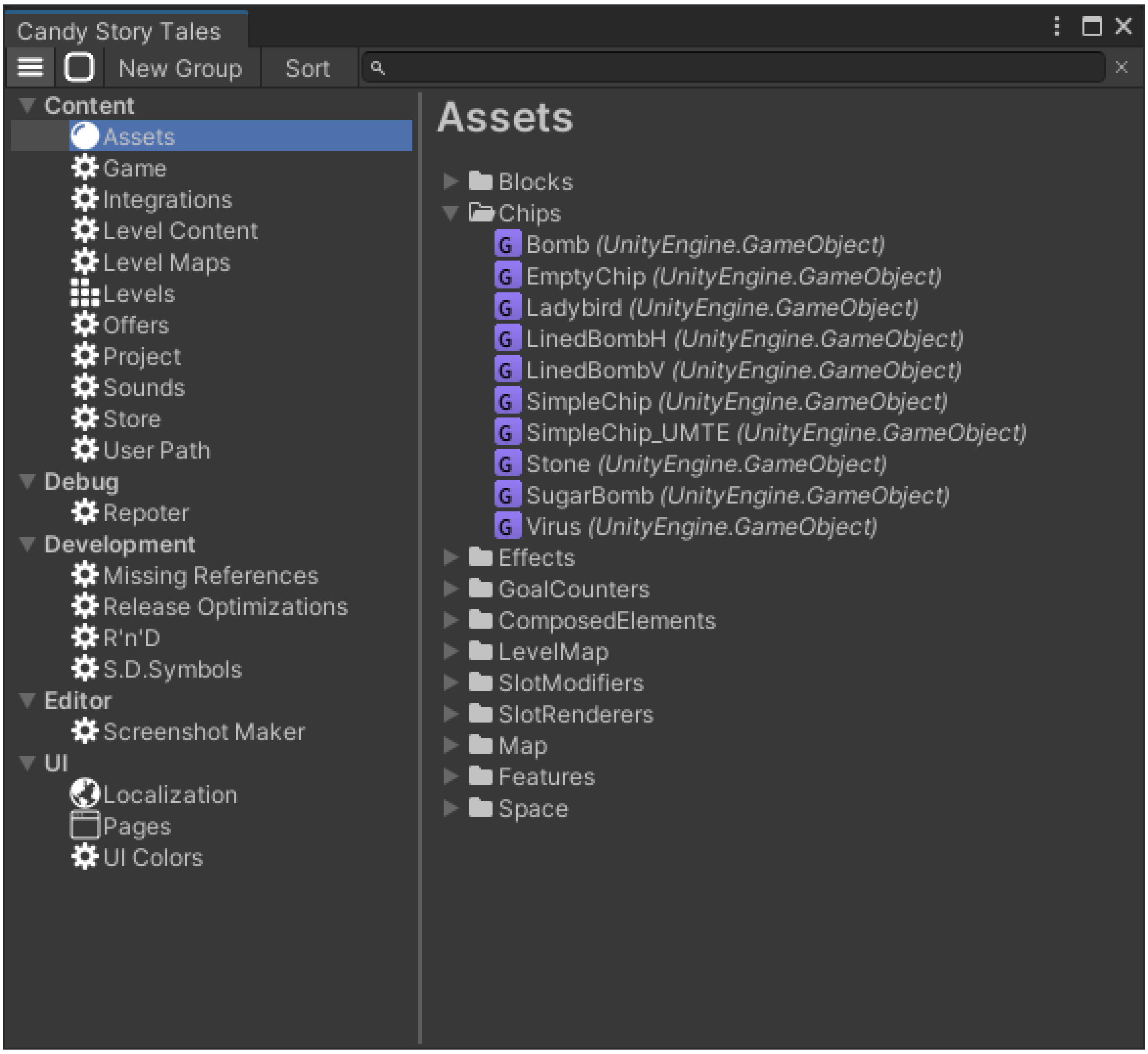
While we play the game, it always need to create different kind of objects. Chips, particle effects, obstacles, etc. But how the game knows what kind of asset to use and when? **Asset Manager** is a pretty simple tool that solves this task. It allows us to collect all necessary assets in one place and to have possibility easy to use them in the code. It the same time using this tool we always see full list of assets we use and also can to edit them fast.



All the assets in the **Asset Manager** must have unique names, because their names are being used as unique keys for getting access. Also there can be not only prefabs, it also can to contain sprites, textures, materials and any other asset types. But only prefab assets can be used for emitting new instances. There is also a special **MonoBehaivour** component that is designed to be used in the Asset Manager. It is called **ContextedBehaviour** - instances of this component (and all inheritors) have a reference to original assets, they can be easy found runtime because they are put into special context and have more other features.

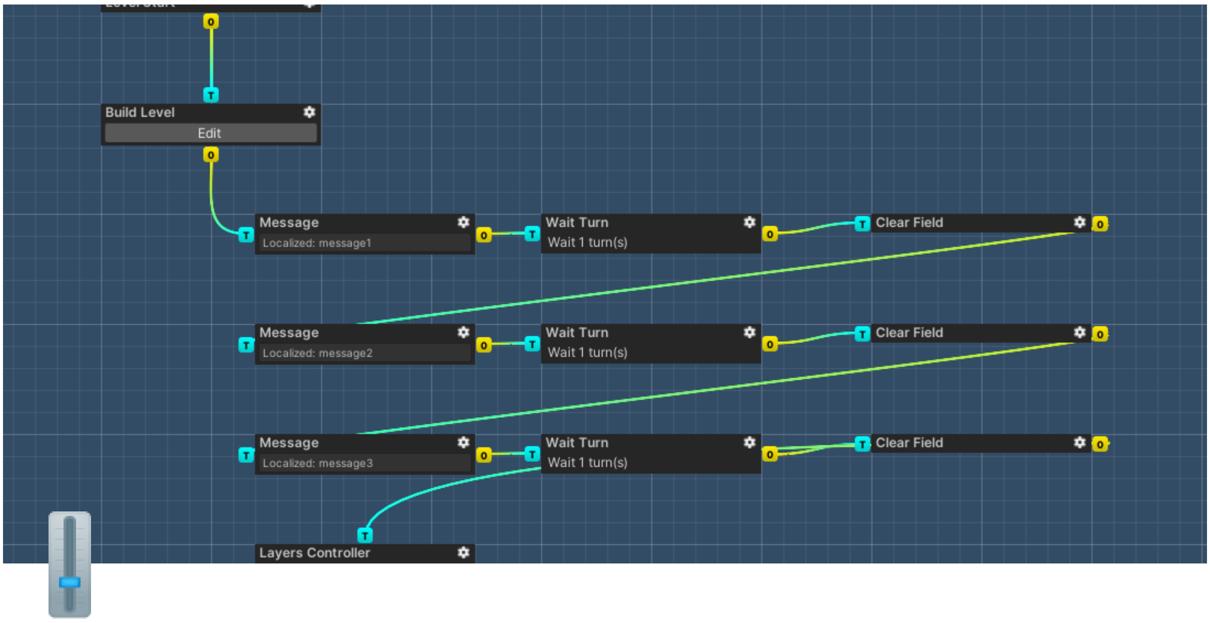
Interface

Lets look how it looks like in Unity. Just open the 🎛 Dashboard and click on **Assets** tab.


Candy Story Tales
New Group
Sort
Content
Assets
Game
Integrations
Level Content
Level Maps
Levels
Offers
Project
Sounds
Store
User Path
Debug
Repoter
Development
Missing References
Release Optimizations
R'n'D
S.D.Symbols
Editor
Screenshot Maker
UI
Localization
Pages
UI Colors
Assets
Blocks
Chips
G Bomb *(UnityEngine.GameObject)*
G EmptyChip *(UnityEngine.GameObject)*
G Ladybird *(UnityEngine.GameObject)*
G LinedBombH *(UnityEngine.GameObject)*
G LinedBombV *(UnityEngine.GameObject)*
G SimpleChip *(UnityEngine.GameObject)*
G SimpleChip_UMTE *(UnityEngine.GameObject)*
G Stone *(UnityEngine.GameObject)*
G SugarBomb *(UnityEngine.GameObject)*
G Virus *(UnityEngine.GameObject)*
Effects
GoalCounters
ComposedElements
LevelMap
SlotModifiers
SlotRenderers
Map
Features
Space

The interface is very simple: there is toolbar with few buttons and search field. The buttons allows to create new asset folders and sort assets by name. And the biggest part of the view is the asset list. All the elements in the list are real files. As for the groups (folders), they are made just for assets structuring.

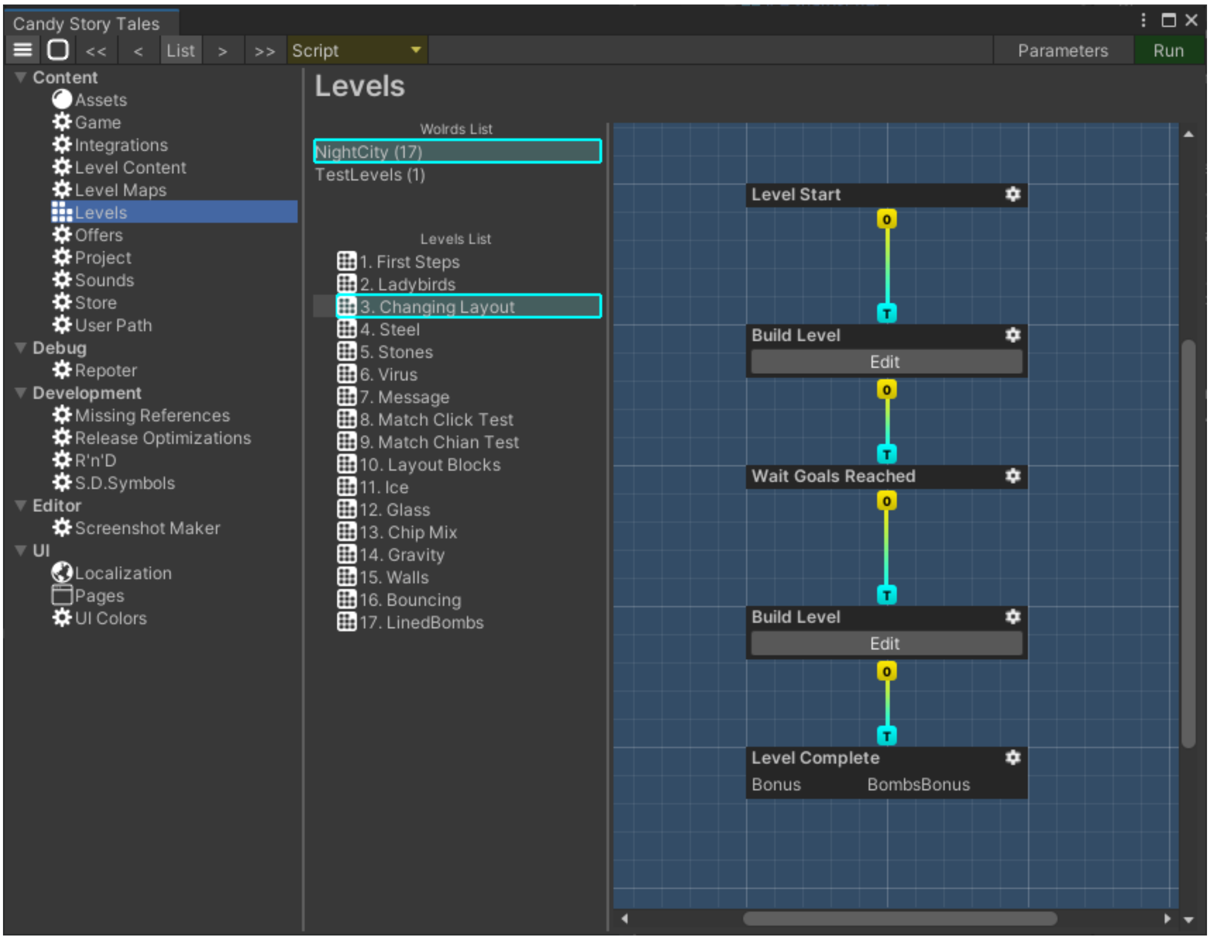
All the assets can be easy edited. Just select an asset you want to edit and it will be selected in the **Project** and in the **Inspector** view. New assets can be added by drag and drop. Just move the asset from the **Project** view into this list.



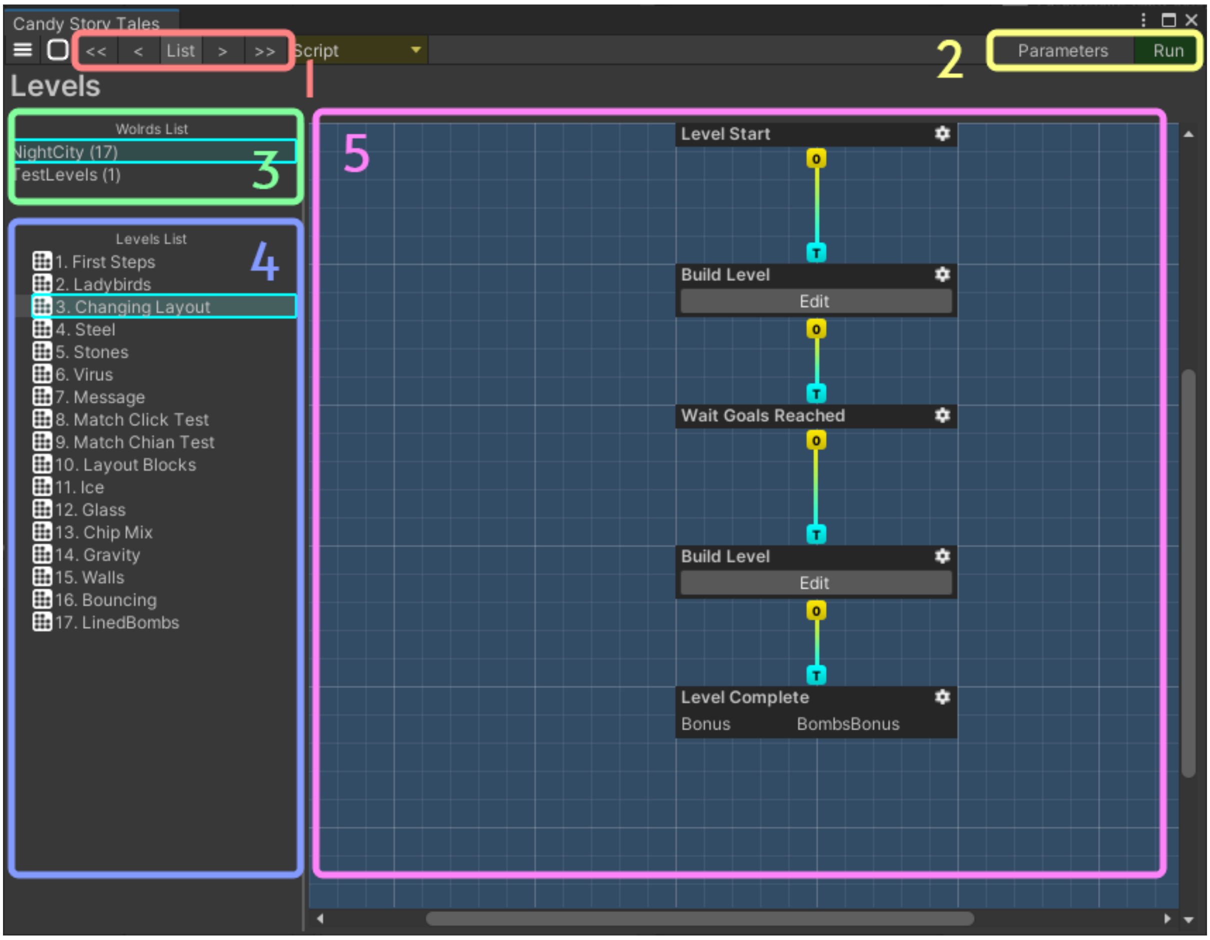
Level Script Editor

Interface

The **Level Editor** can be found in the Dashboard in **Levels** tab.



There is 5 main elements of the editors UI.



1. **Navigation Bar —** five buttons which allows to select *first, previous, next* and *last* levels in the selected world. There is also **List** button, that allows to hide/show the **Levels list**.

2. **Toolbar** — it contains two useful buttons. **Parameters** button opens a window with level parameters. The second one is **Run** button — it allows to start the selected level fast.

3. **Worlds List** — the list of all level worlds from the project. You can have several level worlds and player will be able to play them with separate progress for each of the worlds. Here you can create worlds, remove, rename and switch them. Current world always highlighted with the blue rectangle.

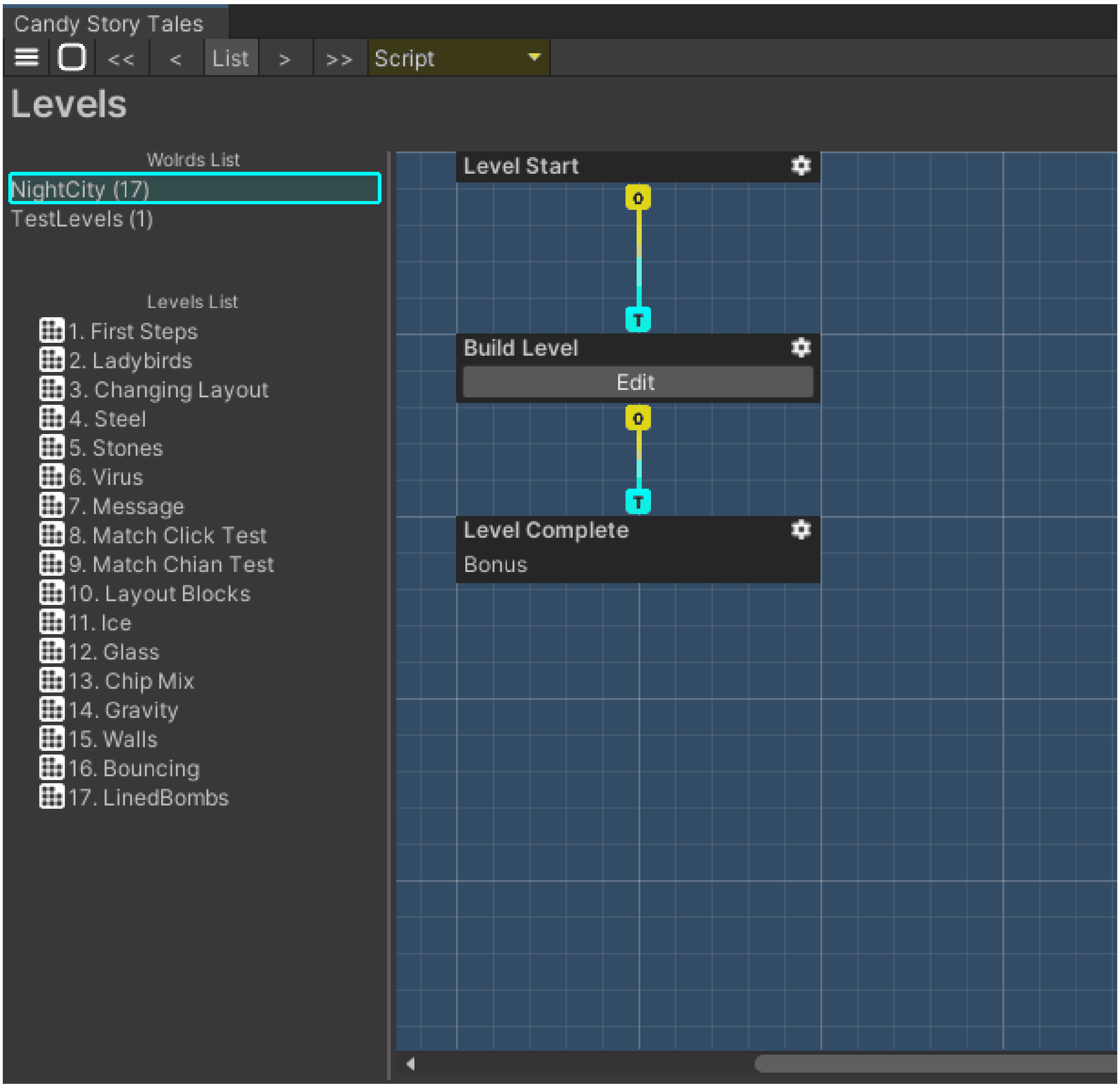
4. **Levels List** — the list of all levels from the selected world. Here you can to create, sort, delete, duplicate, group levels. Selected level always highlighted with the blue rectangle.

5. **Level Script Node Editor** — the node editor of the selected level. It uses the 🌿 YNodes system for editing.

Creating New Level

To create new level let's select the world where you want to create it. Currently we have only two worlds: **Night City** and **Test Levels**. First one is used for public released content, but the second one for testing.

Then by left mouse button click create a new level with one of presets. The **Default** preset is recommended, because it works out of the box.



Wolrds List

NightCity (17)
TestLevels (1)

Levels List

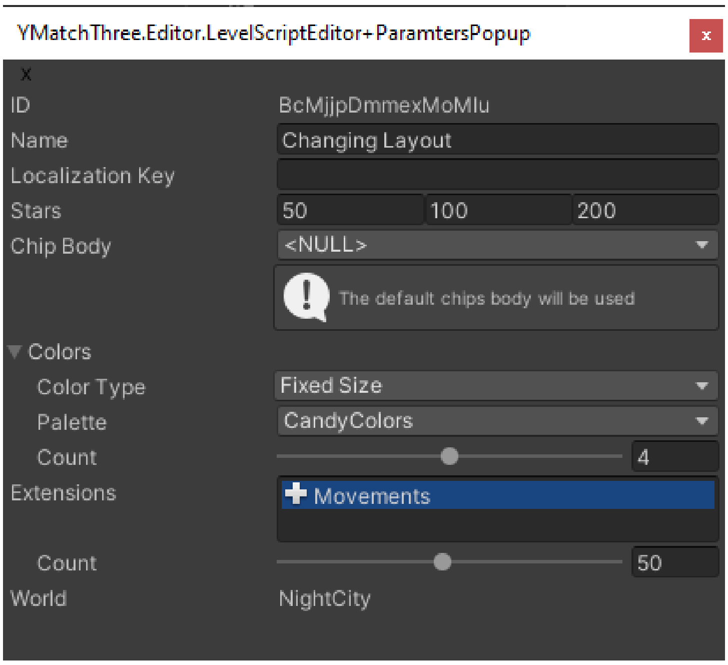
1. First Steps
2. Ladybirds
3. Changing Layout
4. Steel
5. Stones
6. Virus
7. Message
8. Match Click Test
9. Match Chian Test
10. Layout Blocks
11. Ice
12. Glass
13. Chip Mix
14. Gravity
15. Walls
16. Bouncing
17. LinedBombs

You also can use any of existed levels as a preset. Just duplicate it and the copy will be added at the end of the levels list.

All the levels are stored in **Streaming Assets** folder by the next path: *Assets / Streaming Assets / LevelScriptOrdered /.*

The first recommended step of the level editing is to open level parameters and setup them. It can be opened using **Parameters** button on the level editors toolbar.

In the **Parameters** window the next fields are located.

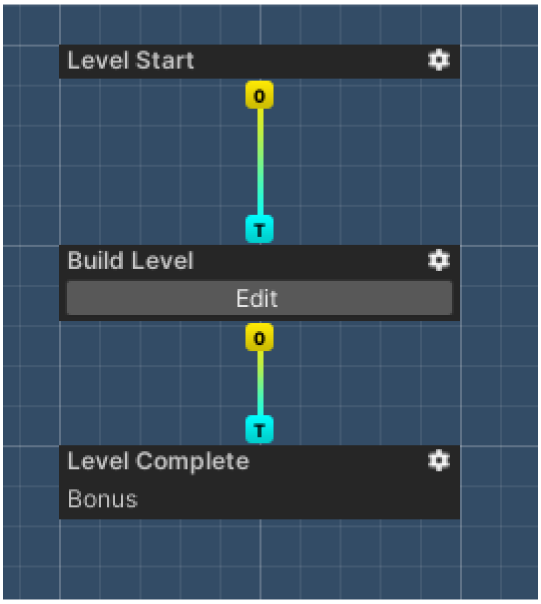


YMatchThree.Editor.LevelScriptEditor+ParamtersPopup

ID	BcMjjpDmmexMoMlu		
Name	Changing Layout		
Localization Key			
Stars	50	100	200
Chip Body	<NULL>		
	The default chips body will be used		
Colors			
Color Type	Fixed Size		
Palette	CandyColors		
Count	4		
Extensions	Movements		
Count	50		
World	NightCity		

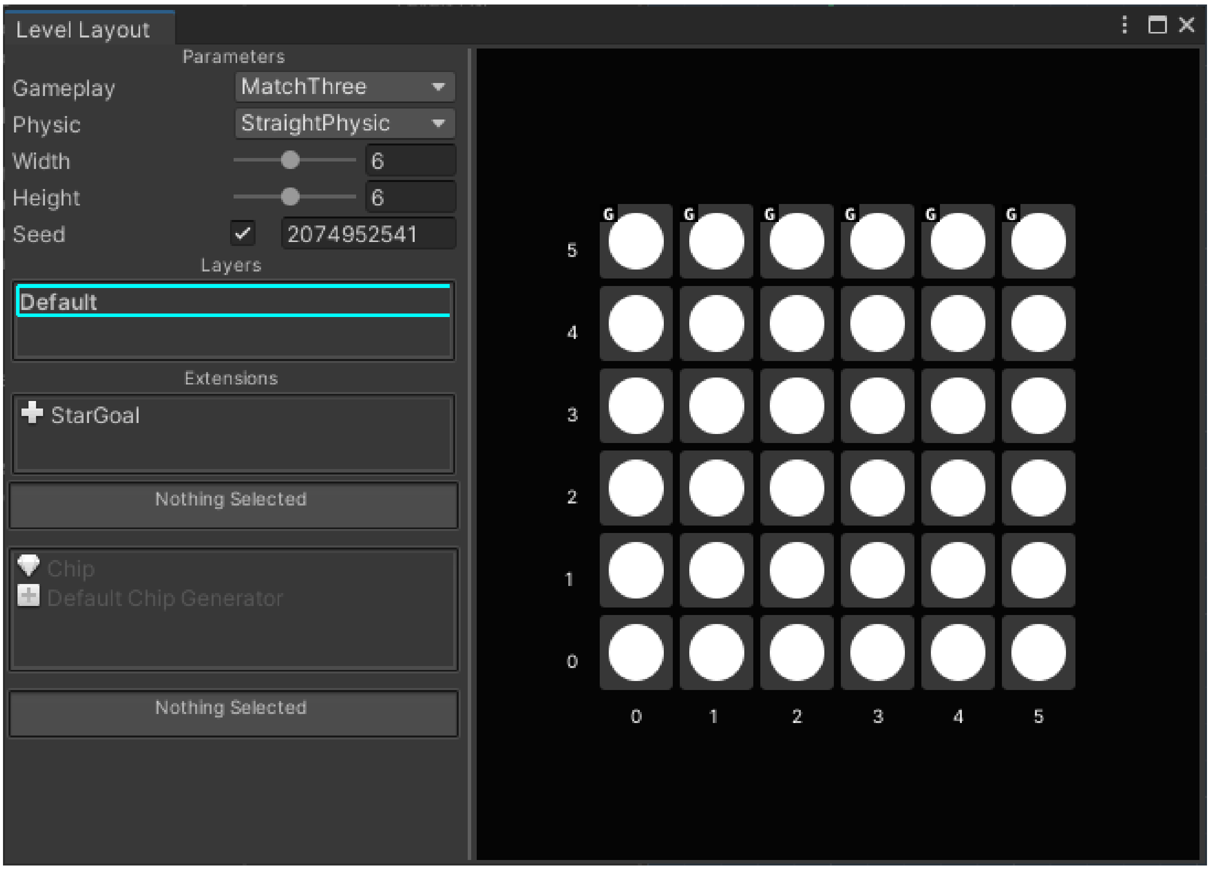
- **ID** — the unique level identifier. It allows you to distinguish one level from another.
- **Name** — the name of the level. Players will not see this name, so you can use it for as you want or not use at all.
- **Localization Key** — the unique key that is used for generating localization entries. Leave it empty, if the level doesn't contain any text messages.
- **Stars** — three integer fields, which contain score point counts for each of the level stars.
- **Chip Body** — the name of default chip body. Leave it empty, if you want to use default body.
- **Colors** — the level color settings. Here you can select one of presets, the color palette, count of colors or colors itself.
- **Extensions** — here can be added and setup level extensions. Currently there is only one extension, which allows to limit count on moves in the level — you can set it up or even remove, if you want to have infinity count of moves in the level.
- **World** — the name of the world, where the level is located.

Now let's look how the **Default** level preset is looks like. It contains the minimum number of nodes, that is necessary for workable level.



The **Level Start** node is launcher node which runs the level script. It doesn't have any settings and only one output port.

The next major node is the **Build Level.** It is the most difficult node in the system. It contains all the settings of the level field such as chip settings, slots, goals, etc. Click **Edit** button if you want to see the **Level Layout** editor, which will be opened in a separate window.


Level Layout

Parameters

Gameplay: MatchThree

Physic: StraightPhysic

Width: 6

Height: 6

Seed: ☑ 2074952541

Layers

Default

Extensions

StarGoal

Nothing Selected

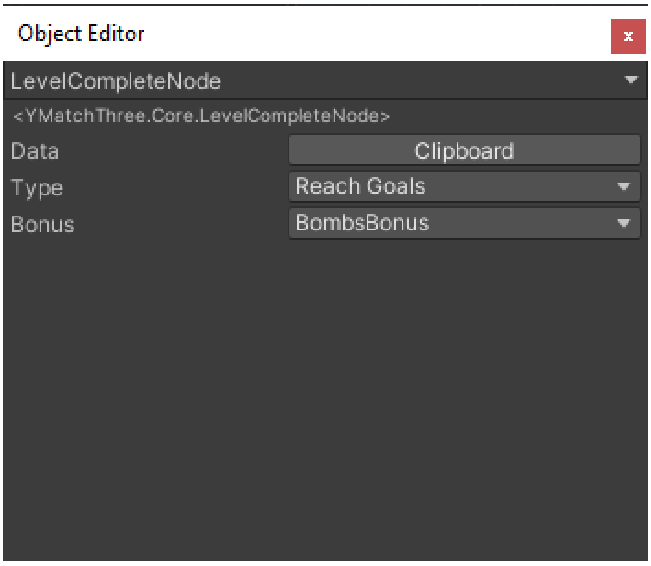
Chip

Default Chip Generator

Nothing Selected

Read more about the **Level Layout** editor in the next tutorial: 🏰 Level Layout Editor

As soon as the level field is created, the **Build Level** node stops to work and runs the next node. The next and the last node in our scheme is the **Level Complete** node. It receives the field from previous node and wait while it is not complete. And then it runs level complete bonus. Open the node settings to set it up. There you can select what it must to wait and what kind of complete bonus to use.


Object Editor

LevelCompleteNode

<YMatchThree.Core.LevelCompleteNode>

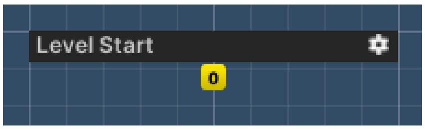
Data: Clipboard

Type: Reach Goals

Bonus: BombsBonus

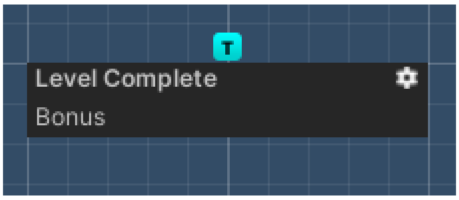
Level Script Nodes

Level Start



The first node of any levels. Without this node the level can't be launched. It emits void value from the output port O.

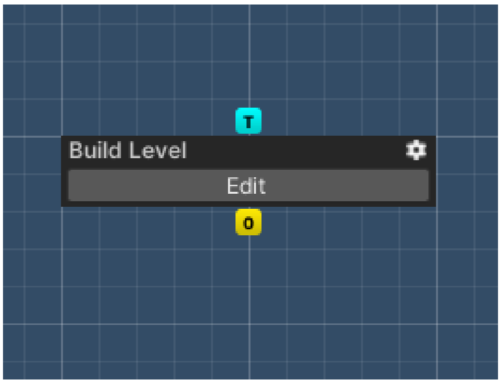
Level Complete



This node is one most important. Without this node any levels can't be finished. So it wait while the goals will be reached, emits complete bonus and shows *You Win* / *You Lose* pages. It always receives only **Field** values in the input port T.

Build Level

As it was described above, this node is used for building the level fields. There is a button that opens the 🏰 Level Layout Editor. It has one input port T, that works as a trigger — it will start to work whatever the values it received. But if it receives **Field** value, it will hide and destroy the received field. And as soon the field is created, it will push the **Field** value through the output port O .

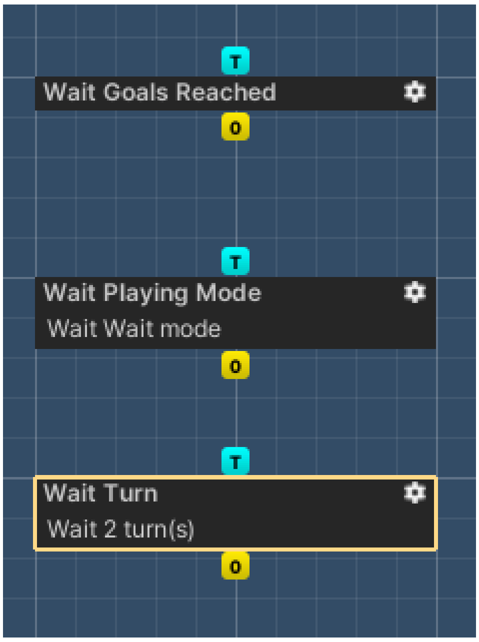


Wait Nodes

There are three very useful nodes, which allows to delay some of other nodes actions.

All the wait nodes receive **Field** value from the input port T , waits while according the logic, and then sends the field through the output port O to the next ports.

The **Wait Goals Reached** node waits while the goals in the field will be reached. This node can be used between two **Build Level** nodes. In this case, when the first field will be complete, it will show the next field. It is very useful, because allows to make levels with a few fields.

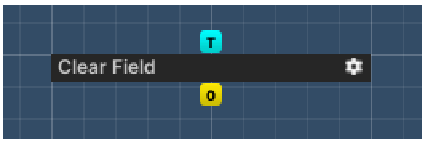


The **Wait Playing Mode** node waits one of the *Playing Modes*. For example, it can be used to wait the "*Matching*" mode, it means that the wait node will not send the field, while it will not start matching the chips.

The **Wait Turn** node allows to wait a few turns to delay the next action. This node will be use in the other node examples below.

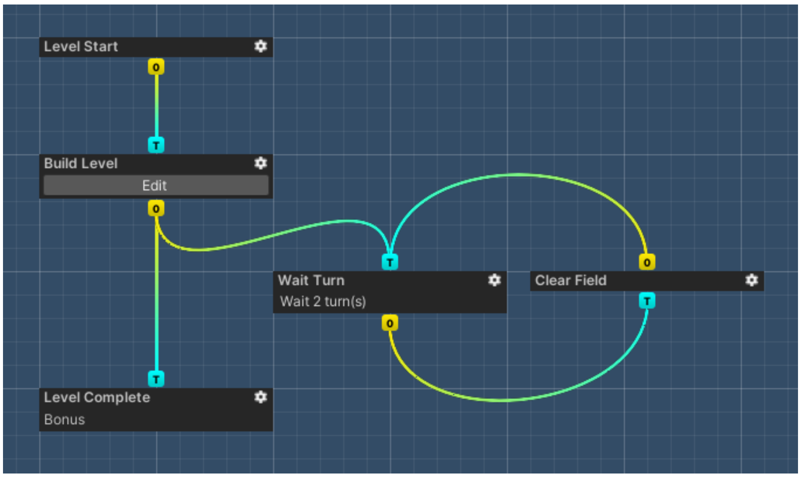
Clear Field

This node receives and send only **Field** values. As soon as it receive a field, it will clear the field from

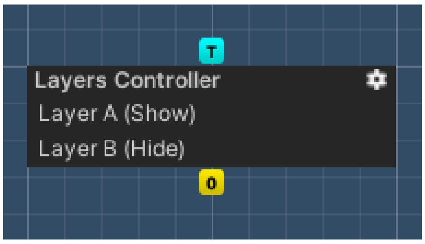


the chips, that it contains.

For example, in the next scheme the field will be cleared from the chips every two turns.



Layers Controller

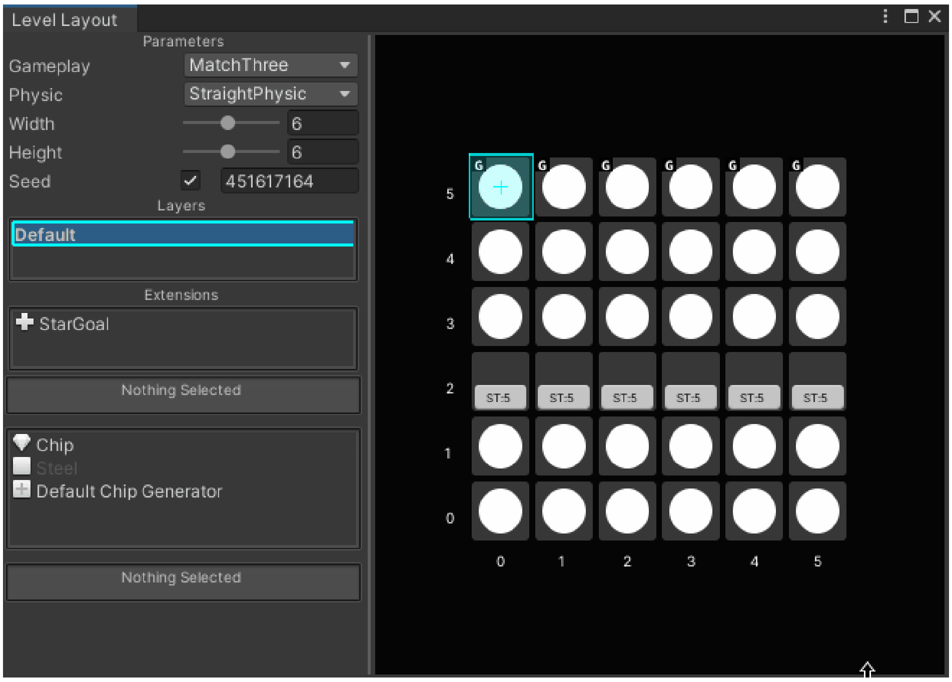


The layer controller is very useful node. It allows to hide and show slots in the current level field.

It receives only **Field** values from the input port T , then applies all layer manipulations, and sends the same Field value from the output port O .

To get it work you need to create a layer in the field that will come in this node. Use the 🏰 Level Layout Editor to do this. Then connect the nodes and setup the **Layers Controller** to operate with this layer.

For example, we have an additional layer of two bottom rows of slots. It is called as *"Bottom"*. And we need to hide it in two turns after the level start, then show again in three turns, and then again start to wait 2 turn to hide it.



Level Layout

Parameters

Gameplay MatchThree

Physic StraightPhysic

Width 6

Height 6

Seed ☑ 451617164

Layers

Default

Extensions

StarGoal

Nothing Selected

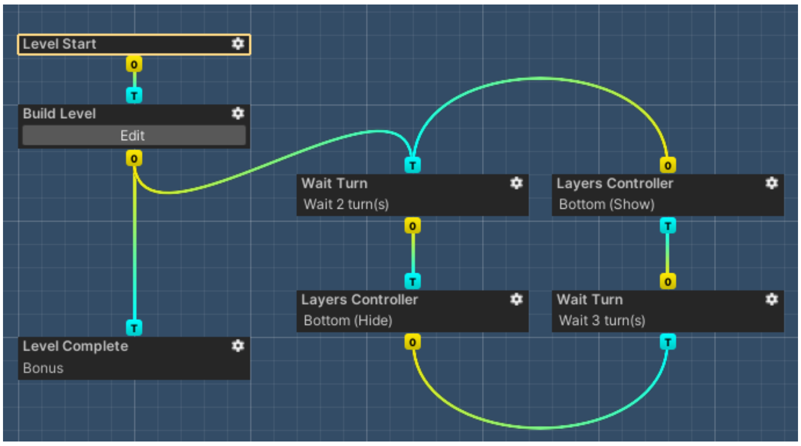
Chip

Steel

Default Chip Generator

Nothing Selected

Of course, in this case we need to use the **Layers Controller** nodes. The scheme of the level script will look like this one:



Level Start

Build Level
Edit

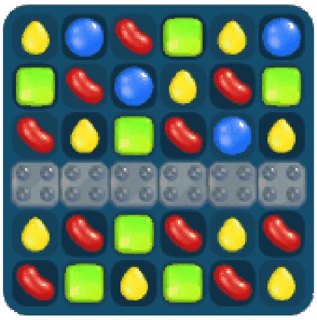
Wait Turn
Wait 2 turn(s)

Layers Controller
Bottom (Show)

Layers Controller
Bottom (Hide)

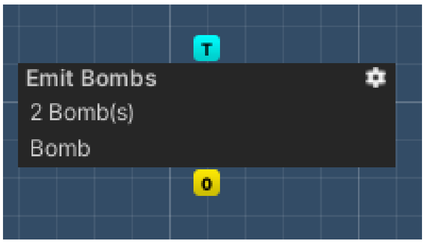
Wait Turn
Wait 3 turn(s)

Level Complete
Bonus



Emit Bombs

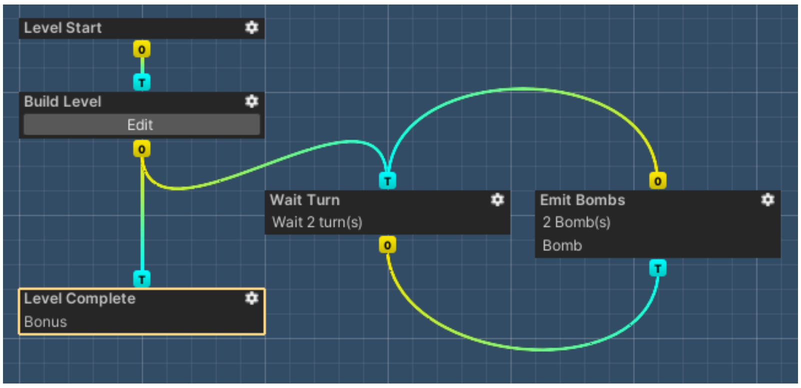
The **Emit Bombs** node allows to create some bombs on the field, that was received from the input port T. The bomb types and count can be setup in the node settings. There is also to use **Slot Tags** to indicate which slots can be used to place them.



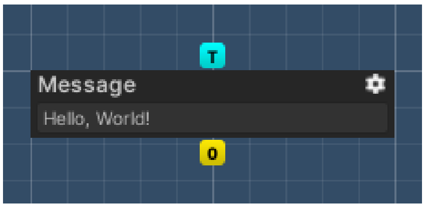
Emit Bombs
2 Bomb(s)
Bomb

Slots Tags is a special slot modifier, that you can place in any slots using 🏰 Level Layout Editor.

As an example, we can emit 2 wrapped bombs each 2 turns using this level script scheme.



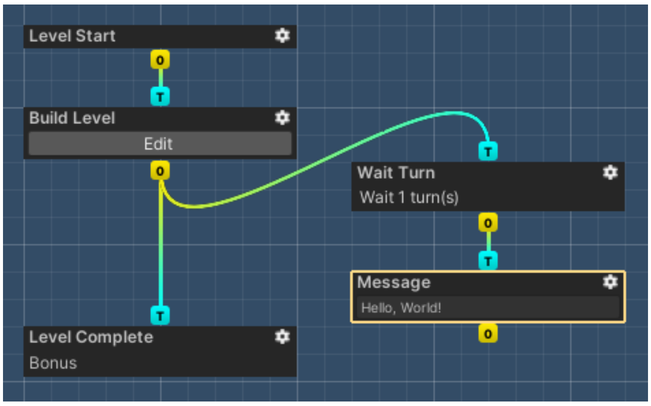
Message



The **Message** node currently is used as an alternative for the 👩‍👧 Dialogue System. It allows to show small text message during the level is playing. That can be useful in tutorials. The text can be provided in the node settings, as well as the condition of its hiding.

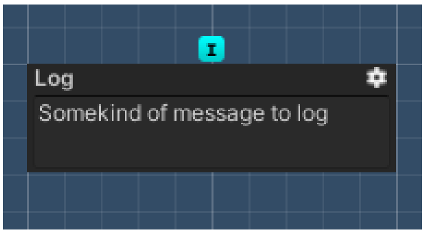
As almost all other nodes, it receives and sends only **Field** values.

Here is a small example of the **Message** node usage:

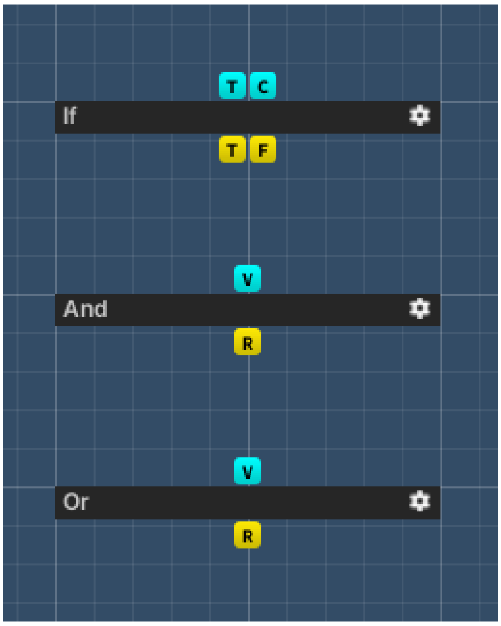


Log

The **Log** node is very useful for debugging the level scripts. It receives any type of values and log in the console the message, the source and the incoming values information.



Boolean Nodes



There are three basic nodes, which is used for boolean operations. But there are no any level script nodes that emits or receive boolean values yet. So in this case these nodes are useless.

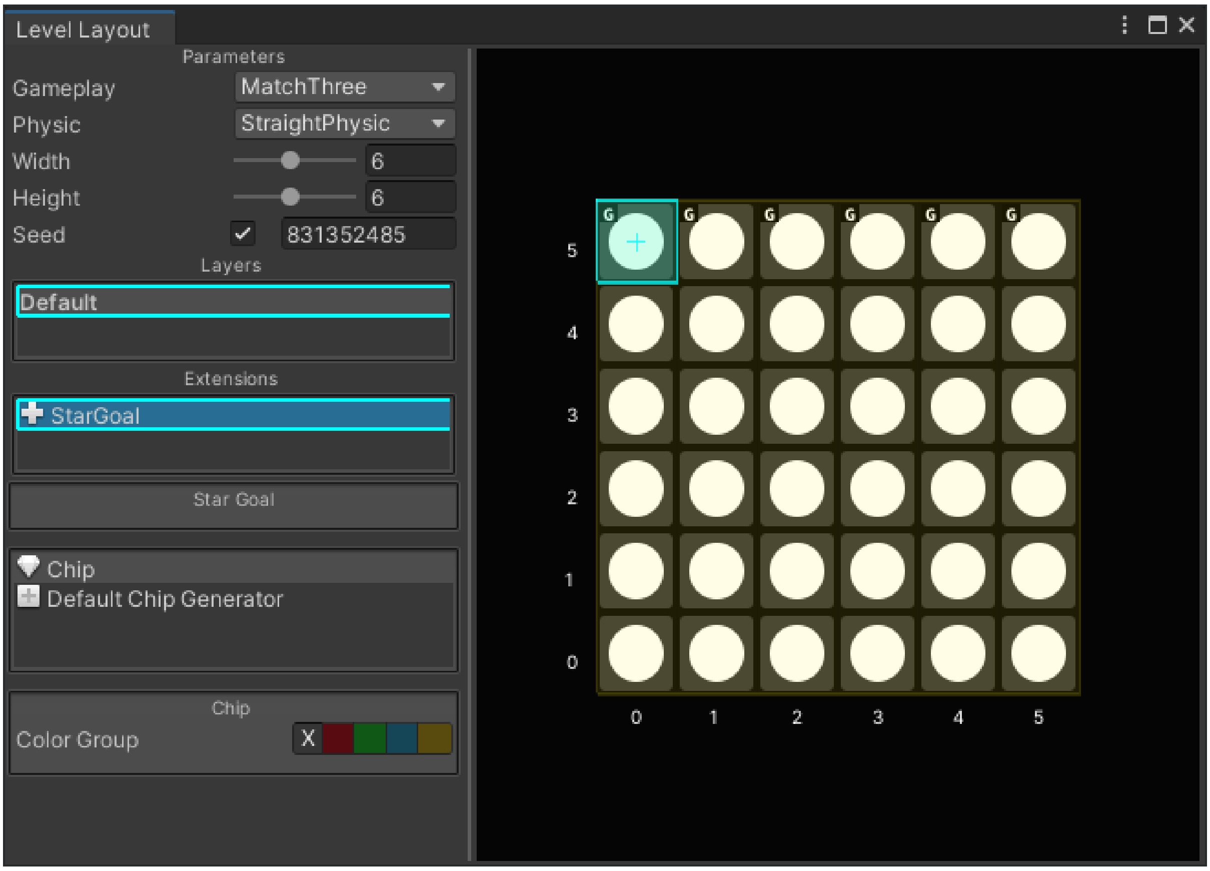
But in a future it will be used for creating non-linear levels. For example, the level start with some of characters speech 🧕 Dialogue System. They ask something to the player and depending on the answer player gets different gameplay (additional bombs, moves, layouts or something similar).

🏰

Level Layout Editor

The **Level Layout Editor** is a part of the Level Script Editor — read this page first.

Now look at the editor. Here we have two important areas.



Level Layout

Parameters

Gameplay: MatchThree

Physic: StraightPhysic

Width: 6

Height: 6

Seed: ☑ 831352485

Layers

Default

Extensions

StarGoal

Star Goal

Chip

Default Chip Generator

Chip

Color Group: X

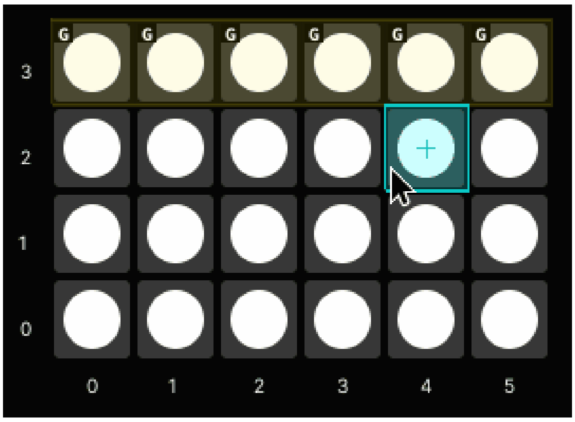
In the left side of the editor we have all general parameters of the level field. The right side contains the layout scheme — here we will select the field content for editing.

Layout View

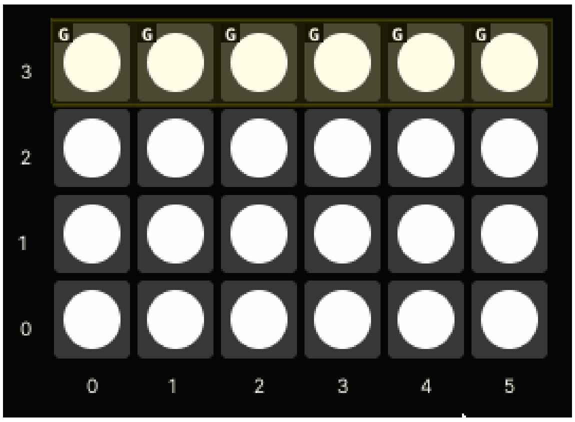
First things first, how to work with the layout view. Here is very simple logic — you select slot coordinates, which you wish to modify, and then do the modifications.

There are few ways to select the slots.

You can to left mouse click one of the slots to select it, but all other slots will be unselected in this case.

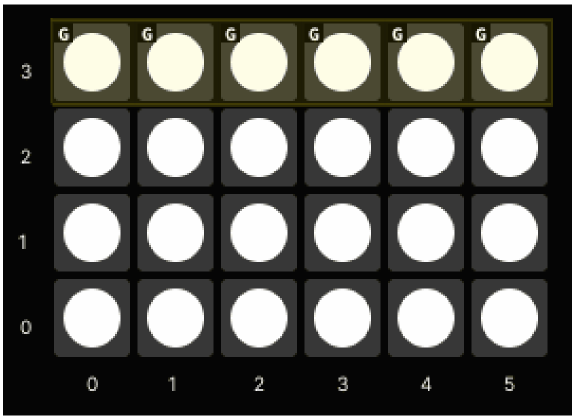


If you need to select few slots in the same time, you need to hold **CTRL/Control** button on your keyboard. And click the slots that must be selected. Repeated click on the selected slot will make it unselected.

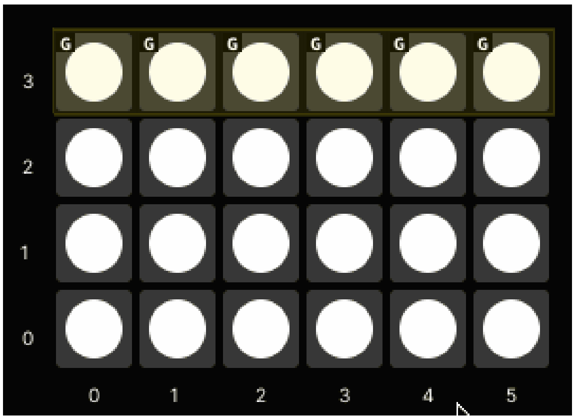


There is also one more key button that allows to select few slots at ones. Hold **SHIFT** key to select line or rectangle area of the slots.

There are no any limitation for selecting. For example, you may select a big rectangle of the slots using **SHIFT** key, and then select a few additional slots using **CTRL** key.

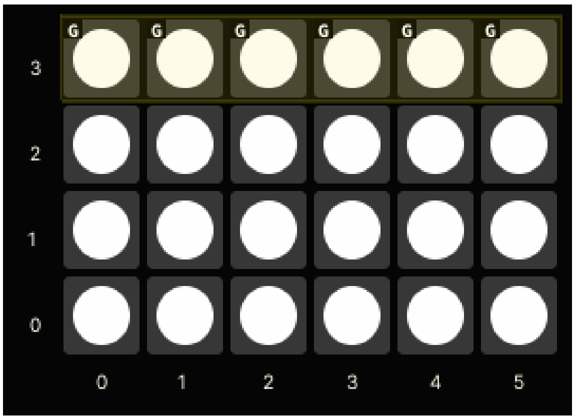


As soon as you select at least one slot, you can right mouse click it to open context menu. It can be used for adding new content in the slots (chips, block and slot modifiers), creating, removing and cleaning them.



There is possibility to add or remove rows and columns right from the layout view. Use **ALT** key to do this. Just hold the key and move the mouse on one of row/columns numbers to remove it. The line will become red, then just left mouse click to remove it.

Move the mouse between numbers to add the line. In this case the line will become green.



Remember that there is a limitation on min and max layout size, so if you will rich it, you will not be able to add or remove lines from the field. And, as you can see on the animation above, the new lines is empty. So you will need to use the context menu to create the slots and fill them with the content, as it was described before.

Parameters View

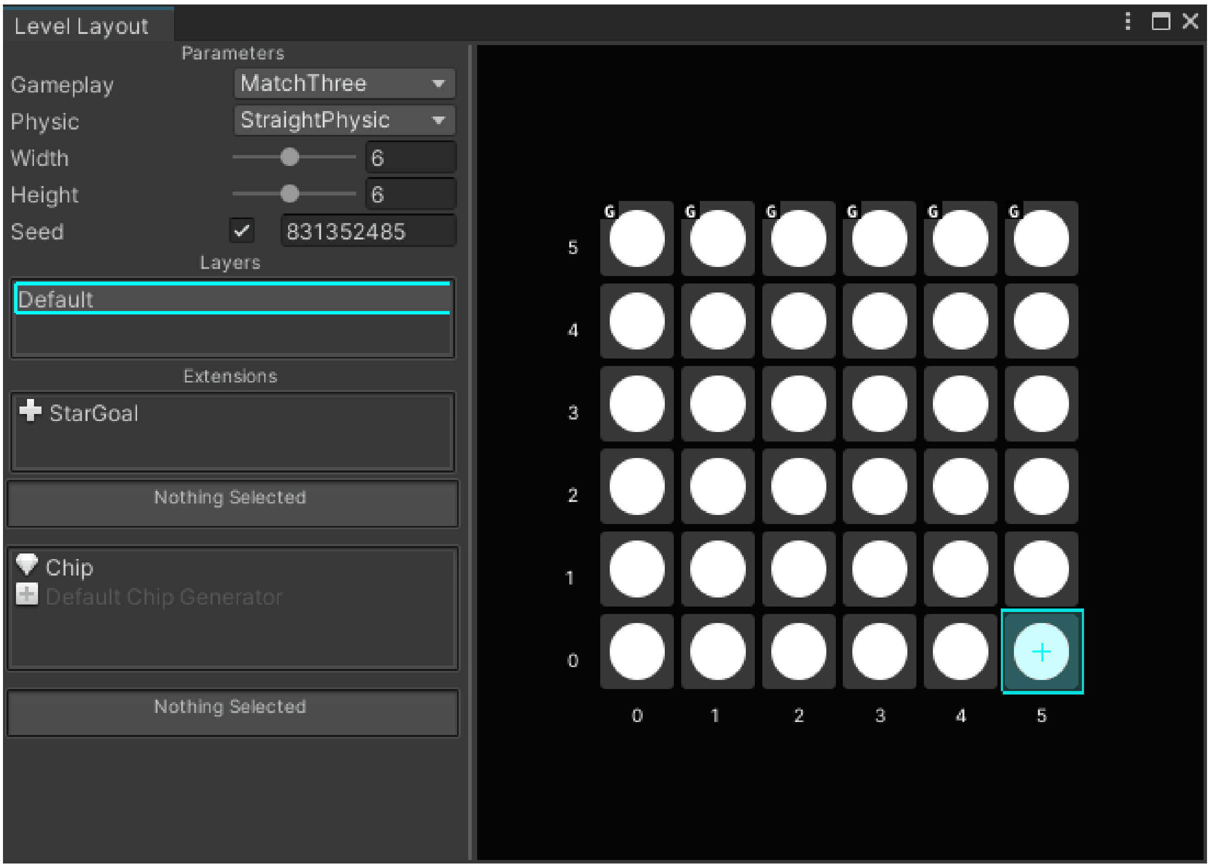
Now let's see what kind of settings we can do in the left side of the editor. The top side of this area contains most general settings:

- **Gameplay** — here we can choose what kind of rules we will use in this field. There are three options now, but you can create new ones in the Level Content editor.
- **Physics** — what kind of chip physics will be used. Here we have two main options: **Realistic** and **Straight**, but you also can create new ones in the **Level Content** editor. It affects on how the chips fall down, they speed, acceleration, etc.
- **Width** and **Height** — allows you to change size of the level layout.

- **Seed** — this option allows to set fixed random seed for the level field. Fields with the fixed seed stops being fully random. So it always generate the same chips as in the first time. It can be very useful in tutorial levels.

There is **Layers** editor area below. It allows to save slot selections as layers and use them for quick selection, showing/hiding the slots in the editor and in the game. Read how to hide and to show slots in the game in the Level Script Editor page.

To create new layer select the slots you want the layer to contain. Then create a new **Selection** layer using context menu.



Level Layout

Parameters

Gameplay: MatchThree

Physic: StraightPhysic

Width: 6

Height: 6

Seed: ☑ 831352485

Layers

Default

Extensions

StarGoal

Nothing Selected

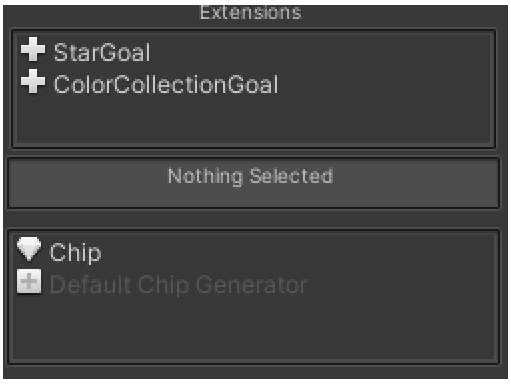
Chip

Default Chip Generator

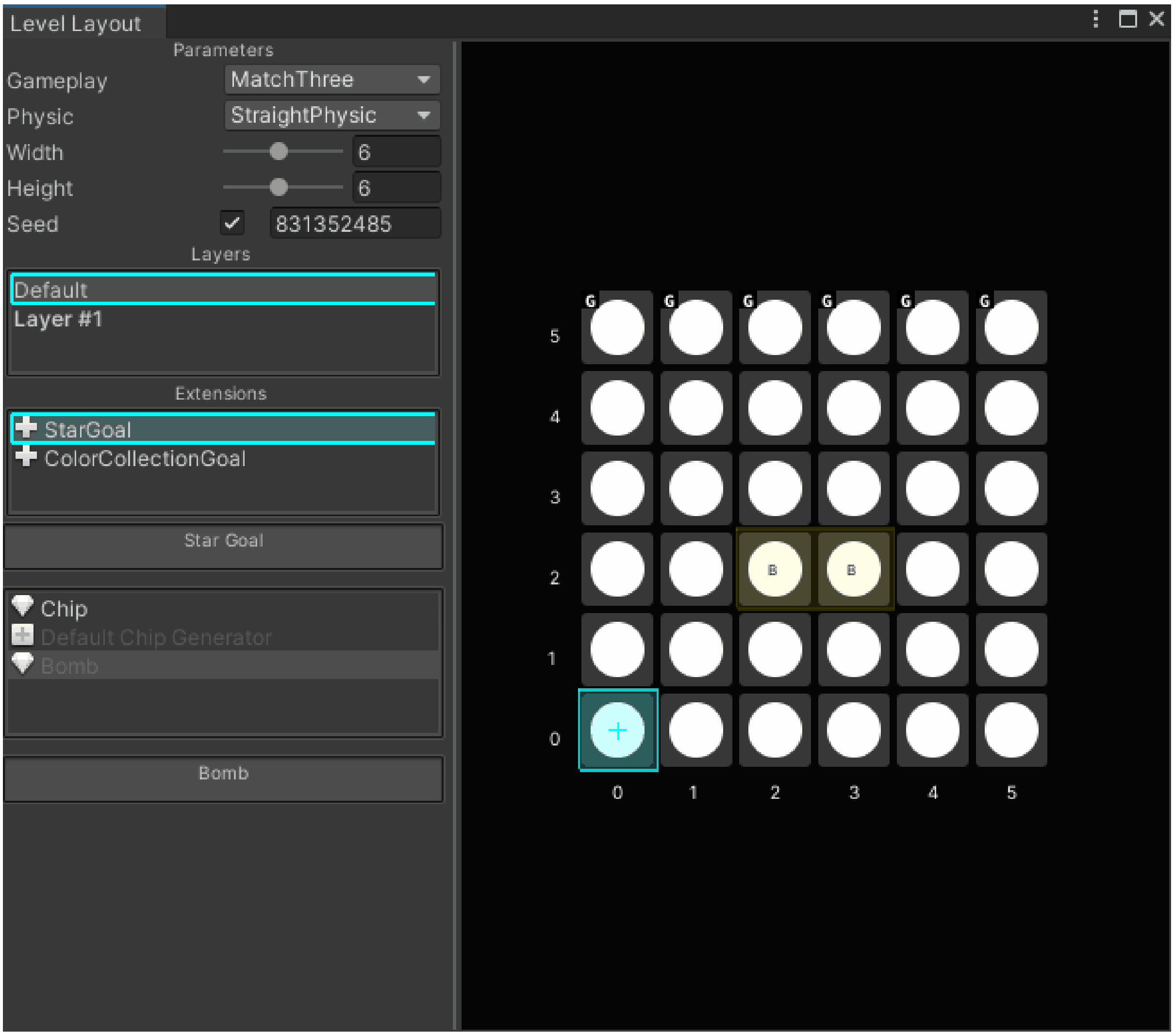
Nothing Selected

The next important area is **Extensions** list. It contains all the level goals and other level extension. So here you can add and setup the level goals. As for the other extension, there is **Gravitation** extension, which allows to modify chips falling directions.

Select an extension to start edit it.

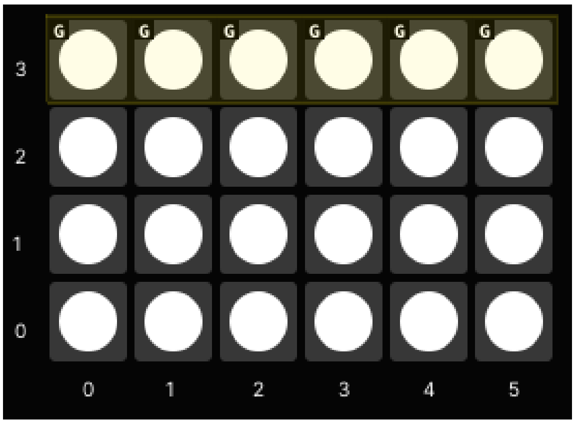


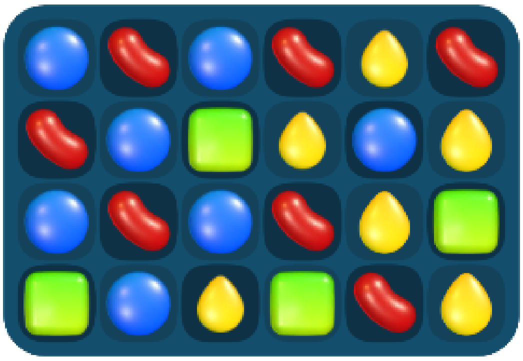
The last part of the left side is the field content list. Here you can see all the features that the field include. Select slots or slot, which you want to edit, and use this list to edit the content from the slots. The gray elements from this list are the content that the field contains, but not included in the selected slots.


Gameplay MatchThree
Physic StraightPhysic
Width 6
Height 6
Seed ✓ 831352485

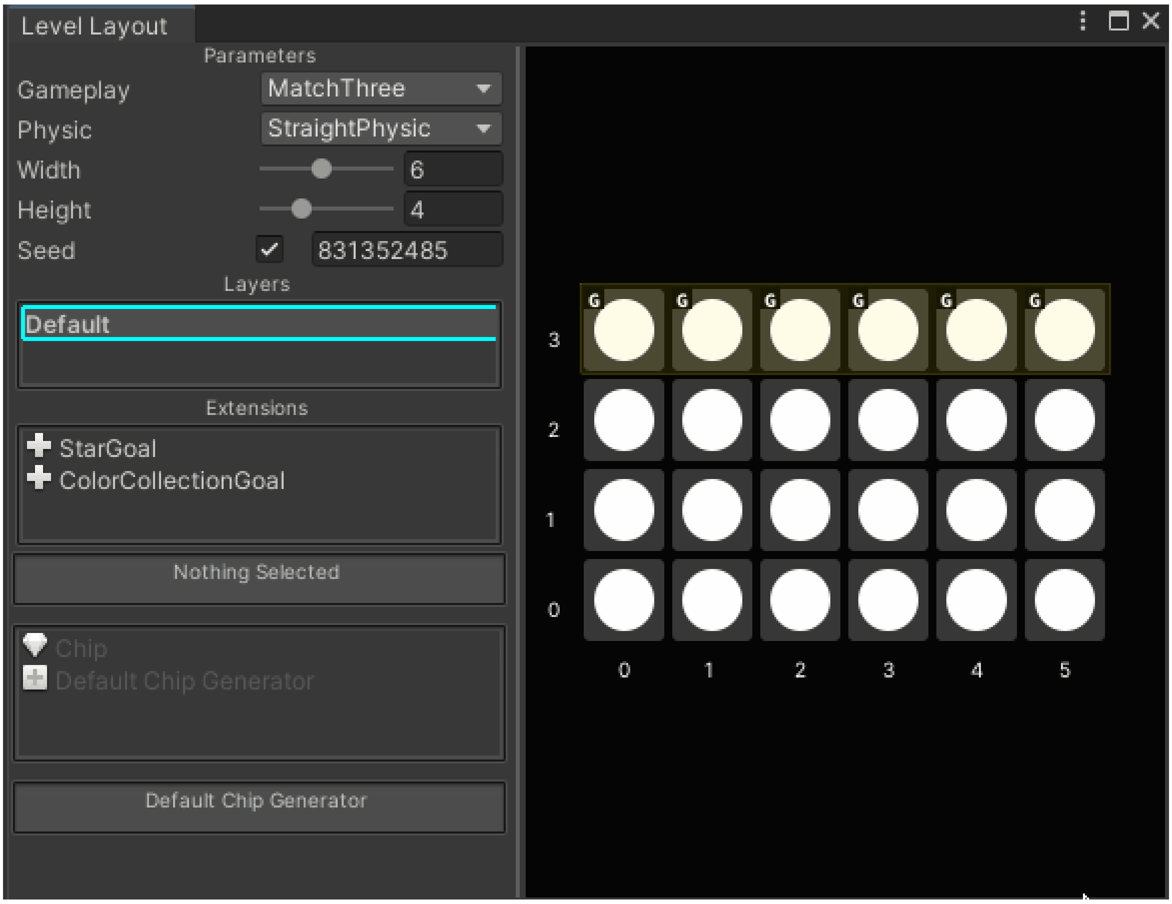
Colors

As you can see we have only white chips the layout view, but they has random colors in the game. This is exactly how colors work.





New chips has always white (random) color by default, and if you want to change it to some certain, first you need to select the slots with the chips you want to edit. Then select the content that you want to edit in the content list and change it to one of the colors.



Level Layout

Parameters

Gameplay: MatchThree

Physic: StraightPhysic

Width: 6

Height: 4

Seed: ☑ 831352485

Layers

Default

Extensions

StarGoal

ColorCollectionGoal

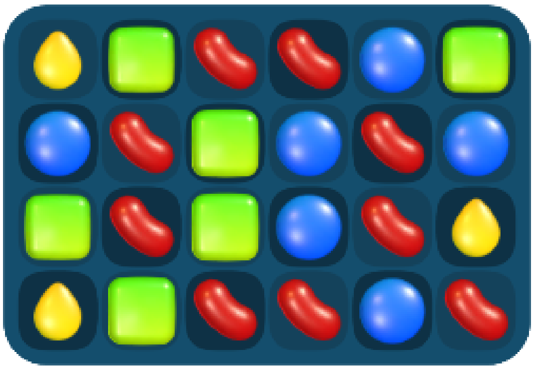
Nothing Selected

Chip

Default Chip Generator

Default Chip Generator

All the chips which is red in the edit are became red in the game. But there is a possibility to shuffle the colors each time the level starts. To do this you need to get back to the Level Script Editor and change the level **Parameters**. You need to change the **Color Type** parameters to **Random**. Or use **Preset** type if you want other color set, but don't want to make the colors random.



Here is a few examples of level generation with **Random** color type. As you can note, we always have the same chips order as we set, but always with different colors.