USER MANUAL

Berry Match-Three Engine v2.1

Contents

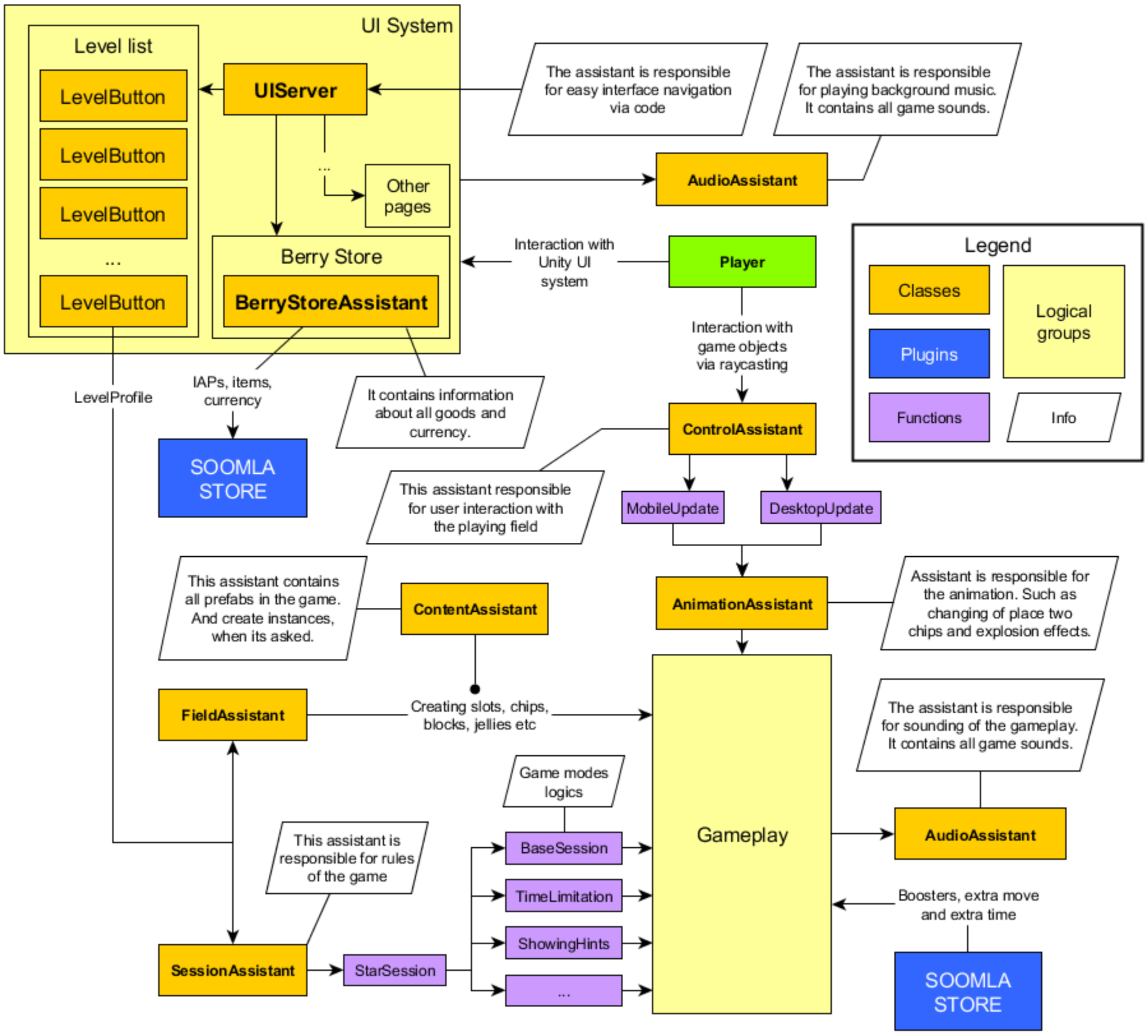
Official store page: http://u3d.as/c3R

Created by Yurov Viktor

email: yurowm+u3d@gmail.com

skype: yurov_viktor

THE PROGRAM STRUCTURE



UI System
Level list
LevelButton
LevelButton
LevelButton
...
LevelButton
UIServer
...
Other pages
Berry Store
BerryStoreAssistant
The assistant is responsible for easy interface navigation via code
The assistant is responsible for playing background music. It contains all game sounds.
AudioAssistant
Interaction with Unity UI system
Player
Interaction with game objects via raycasting
Legend
Classes
Logical groups
Plugins
Functions
Info
LevelProfile
IAPs, items, currency
It contains information about all goods and currency.
SOOMLA STORE
ControlAssistant
This assistant responsible for user interaction with the playing field
MobileUpdate
DesktopUpdate
This assistant contains all prefabs in the game. And create instances, when its asked.
ContentAssistant
AnimationAssistant
Assistant is responsible for the animation. Such as changing of place two chips and explosion effects.
FieldAssistant
Creating slots, chips, blocks, jellies etc
The assistant is responsible for sounding of the gameplay. It contains all game sounds.
Game modes logics
Gameplay
AudioAssistant
This assistant is responsible for rules of the game
BaseSession
TimeLimitation
ShowingHints
...
Boosters, extra move and extra time
SessionAssistant
StarSession
SOOMLA STORE

On the picture above, you can see the scheme of the framework structure. The scheme shows 2 main logic groups - user interface and gameplay.

The user interface was implemented on native instruments of Unity engine. However, while Unity engine does not consist page manager, pop-ups and other meaningful structures of interface, special script was made to solve this problem - **UIServer**. If you are interested in work of this script please read the article below.

Interface consists 2 important parts - Level List and Berry Store.

The **Level List** consists the set of game levels. You can create new levels, change old ones, duplicate and sort them. Detailed description follows below.

Each level has a button (UI.Button component), you can push it for writing information about level in memory – **LevelProfile** class.

Then, based on writed information, generation of game field using component **FieldAssistant** is take place. Function *FieldAssistant.main.CreateField().*

The other important component is **ContentAssistant** – contains all prefabs which are used in gameplay. This component give new request instances using functions *ContentAssistant.main.GetItem*.

After generation of game field, logic realizing procedures of the game mode are started. You can make it using *SessionAssistant.main.StartSession(FieldTarget sessionType, Limitation limitationType)* function. Where *sessionType* – is type of the target of game, *limitationType* – type of game limitation (game limited by time or game with limited count of moves).

Function **StartSession** starts 6 coroutines:

1. *BaseSession* – basic coroutine, which take decisions to continue game or stop it when some results are achieved while game session (win or lose)
2. One of coroutines *ScoreSession, JellySession, BlockSession, ColorSession* (depends on argument *sessionType*) – monitors if the session target is achieved while the game session.
3. One of coroutines *TimeLimitation, MovesLimitation* (depends on argument *limitationType*) – monitors if player has enough resources while game session, and if it is not, offers him resources or announce that session is over.
4. *FindingSolutionsRoutine* - coroutine that continuously searching for chip combinations on the game field. And if it found one, these combinations would be destroyed and player get score points.
5. *ShowingHintRoutine* – coroutine that searching for different

potential moves and if player is taking decision for too long, these coroutine flashes one of the possible moves.

6. *ShuffleRoutine* – that also searching for potential moves. If there is a situation that no one move left, then these coroutine would mix chips till at least one possible move will appear.

The component **ControlAssistant** also plays significant role in gameplay. It takes responsibility for interaction between player and field objects. It considers the platform specifics and use different work regimes on PC and mobile gadgets.

The main task of this component is to define on which chip the player pressed and on which side the player moved this chip. Based on this information the function *AnimationAssistant.main.SwapTwoItem (Chip a, Chip b)*, is starts. Where *a* - chip on which player pressed and *b* - chip with which should change places.

AnimationAssistant - takes responsibility for animations. On other words, the changing places of 2 chips – is it's main task. There is also *Explode* function that realizes the effect of chips bounce if some explosion take place.

While moving 2 chips this function seeks advice of **SessionAssistant** class.

SessionAssistant class takes responsibility for the logic of game. It searches possible options, shows hints, mixing chips, when no potential moves left, it realizes the logic of game mods, analyses combinations on the game fields and so on.

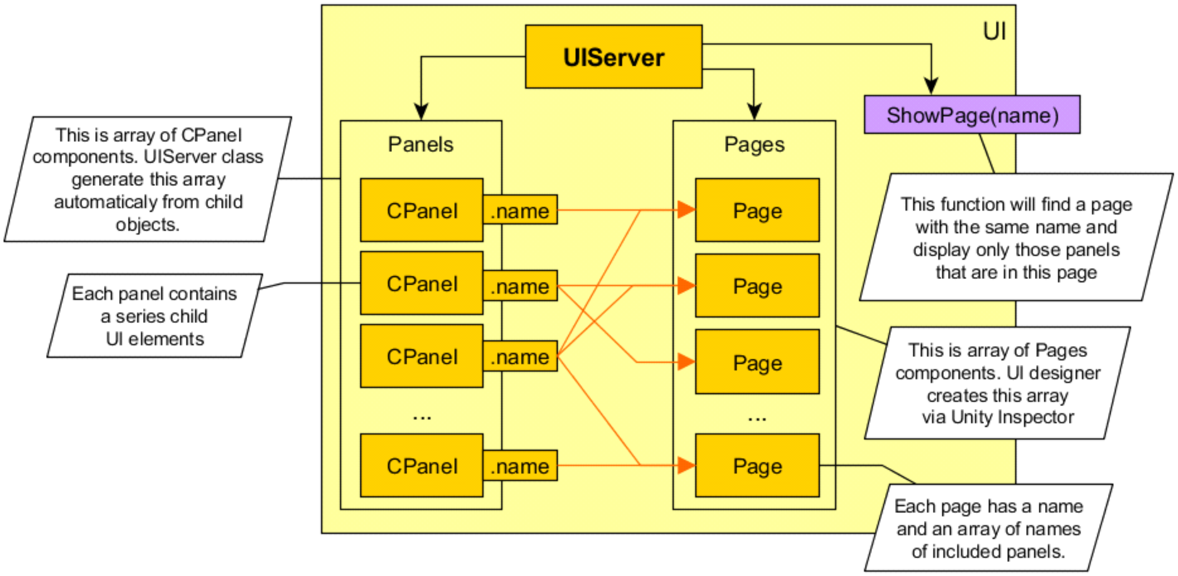
Also we should notice about the **AudioAssistant** class - it consists sound base and music for game. When it is necessary to play some sound, functions *AudioAssistant.main.Shot(string clip)* – (for starting of certain sound) and *AudioAssistant.main.PlayMusic(string track)* – (for changing music background) are using to appeal to **AudioAssistant** class.

All assistents in demo-scene have common GameObject. It names **Core** and it is located in the root of scene hierarchy.

UIServer – PAGE MANAGER

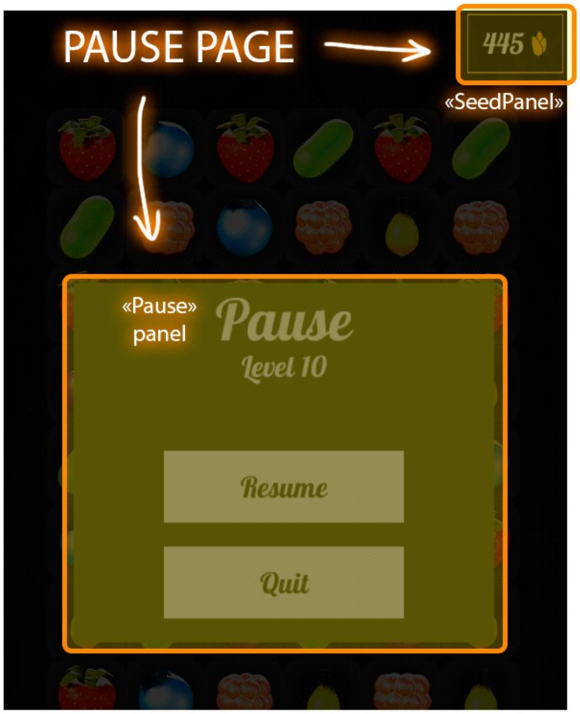
The Game structure is made in the way that for different tasks different classes are reserved. As you can see, **UIServer** works as page manager. There are also many auxiliary components that located in the folder: *Yurowm / Match-Three Engine / Scripts / Interface / UI.* These components were made for unidirectional tasks. For example, for task to show the number of level on pop-ups or to show the time when playing the game limited by time.

Component **UIServer** is quite simple in using.

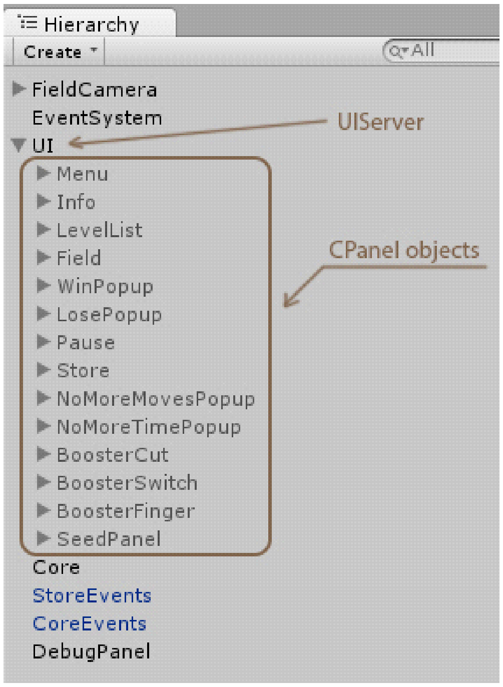

UI
UIServer
ShowPage(name)
Panels
Pages
CPanel
.name
CPanel
.name
CPanel
.name
...
CPanel
.name
Page
Page
Page
...
Page
This is array of CPanel components. UIServer class generate this array automaticaly from child objects.
Each panel contains a series child UI elements
This function will find a page with the same name and display only those panels that are in this page
This is array of Pages components. UI designer creates this array via Unity Inspector
Each page has a name and an array of names of included panels.

On the scheme above you can see that component uses 2 definitions – panel and page.

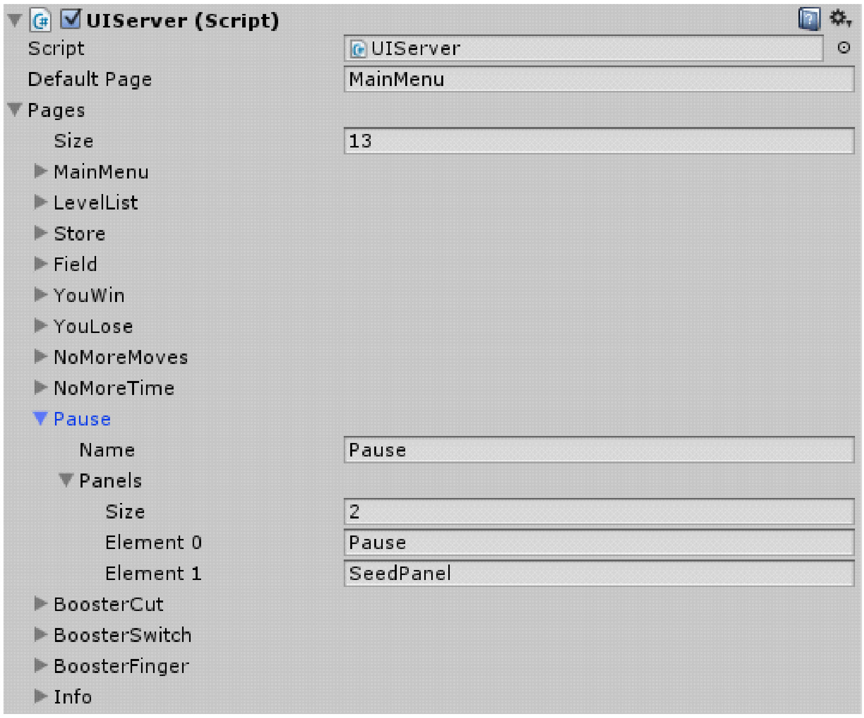
Panel – is the part of interface that consists of interface elements. Panel could describe the whole page or some part of the page as well. Generally in game all pages contains one panel. However the page Pause consists of 2 panels (see the example below):



All panels that you want to use in game should be child objects of the **UIServer** component. Each panel should have at least **CPanel** component. That is why **CPanel** component is necessary to initialize object as panel. The list of panels in demo-scene you can see below:



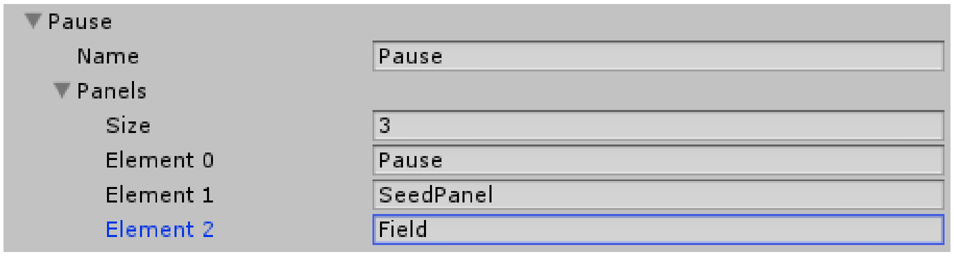
Information about pages is completed in **UIServer** component inspector.



```
UIServer (Script)
  Script              UIServer
  Default Page        MainMenu
Pages
    Size              13
  MainMenu
  LevelList
  Store
  Field
  YouWin
  YouLose
  NoMoreMoves
  NoMoreTime
  Pause
      Name            Pause
    Panels
        Size          2
        Element 0     Pause
        Element 1     SeedPanel
  BoosterCut
  BoosterSwitch
  BoosterFinger
  Info
```

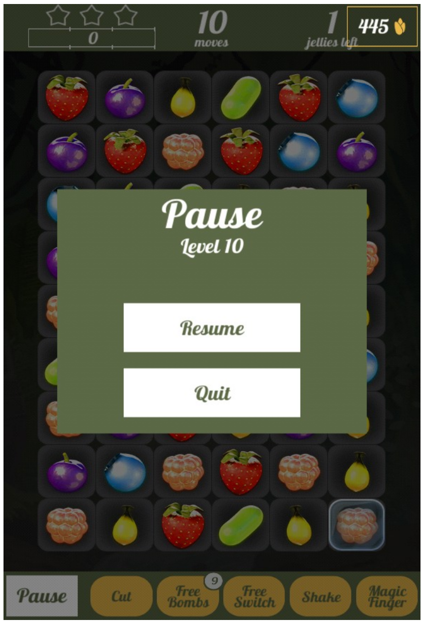
Pages are separate elements of the "Pages" array. Each page has unique name and list of the panel names that this page includes.

As an example, let us add one more panel in Pause page. Let it be interface panel of the game field "Field":



```
Pause
    Name            Pause
  Panels
      Size          3
      Element 0     Pause
      Element 1     SeedPanel
      Element 2     Field
```

Resulting page looks like:



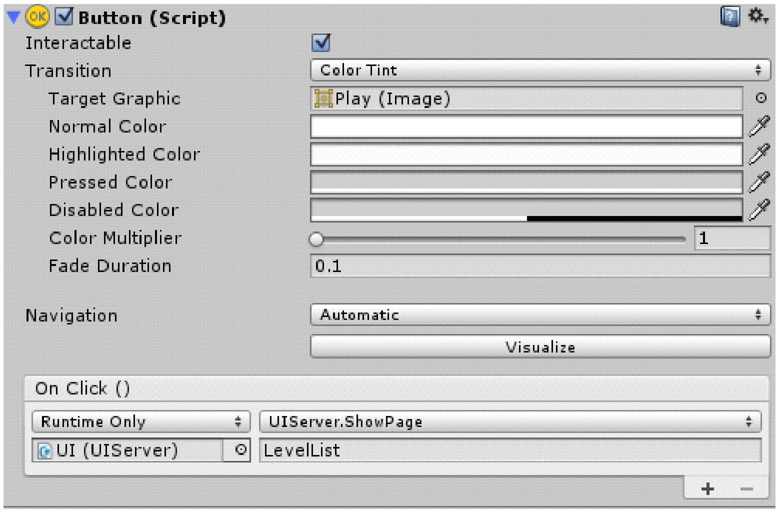
As you can see in the pause menu the interface of the game field is reflects.

The UIServer component also has defaultPage parameter – the page name that opens when player starts a game.

To open some page the function U*IServer.main.ShowPage(string name)* is using. Where name – is the name of page.

There is also function *UIServer.main.ShowPreviousPage()* – it opens the previous page.

It is better to use the event *onClick* in UI.Button component when creating interface.



Button (Script)

Interactable	☑
Transition	Color Tint
Target Graphic	Play (Image)
Normal Color	
Highlighted Color	
Pressed Color	
Disabled Color	
Color Multiplier	1
Fade Duration	0.1
Navigation	Automatic
	Visualize

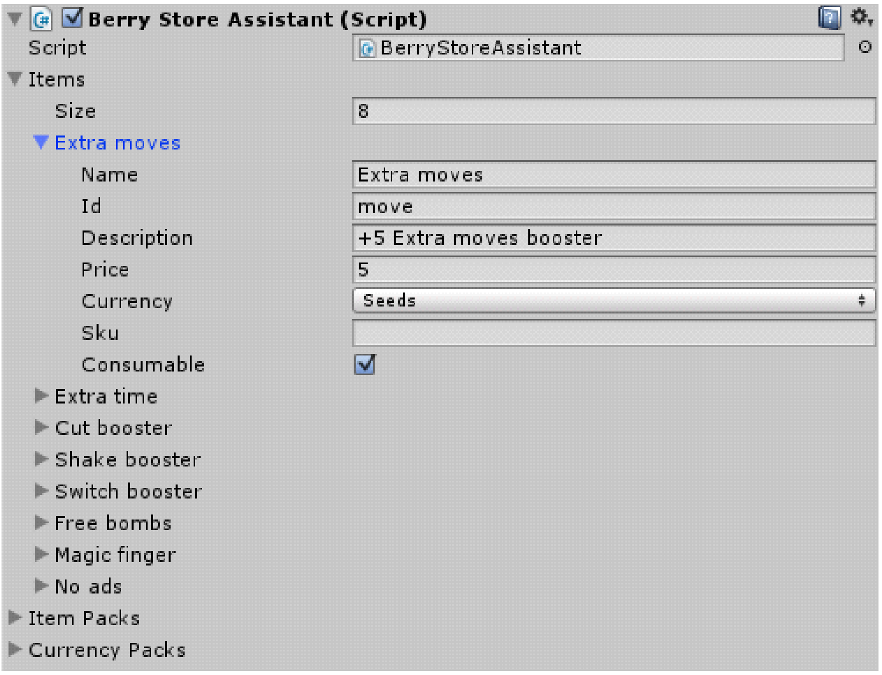
On Click ()

Runtime Only	UIServer.ShowPage
UI (UIServer)	LevelList

BerryStoreAssistant – STORE SETTINGS AND IAP INTEGRATION

BerryStoreAssistant component includes the interface realization of **Soomla Store** plugin. It takes responsibility for interaction between **Soomla Store** plugin. In **Soomla Store** plugin inspector it is possible to input the information about goods – about IAPs or about items that sold for game currency.

The inspector of **Soomla Store** plugin looks like:



▼ ☑ **Berry Store Assistant (Script)**

Script	BerryStoreAssistant
▼ Items	
Size	8
▼ Extra moves	
Name	Extra moves
Id	move
Description	+5 Extra moves booster
Price	5
Currency	Seeds
Sku	
Consumable	☑
▶ Extra time	
▶ Cut booster	
▶ Shake booster	
▶ Switch booster	
▶ Free bombs	
▶ Magic finger	
▶ No ads	
▶ Item Packs	
▶ Currency Packs	

There is a Items array that consists of describing separate goods items objects. Important parameter id - using this identifier the goods would be identified in the game. Price parameter – defines how many items of currency are the price of some item. Currency parameter – defines what kind of currency is used: dollars or seed. If dollars was used then SKU from similar IAP should be noted.

Item Packs array – packs of goods. They have their own pack ID and item ID in pack as well. Item ID in pack is taken from previous array. Item Count – the count of items in pack, price, currency and SKU (if required).

Currency Packs array contains currency packs (seeds). There is only one option to buy for real money, so it is impossible to choose type of currency. At the same time, SKU is required.

The purchase is carried out using special function *BerryStoreAssistant.main.Purchase(string id)*. Asset already contains purchase button component - **BerryStorePurchaser**. It is also using for purchasing in demo-scene.

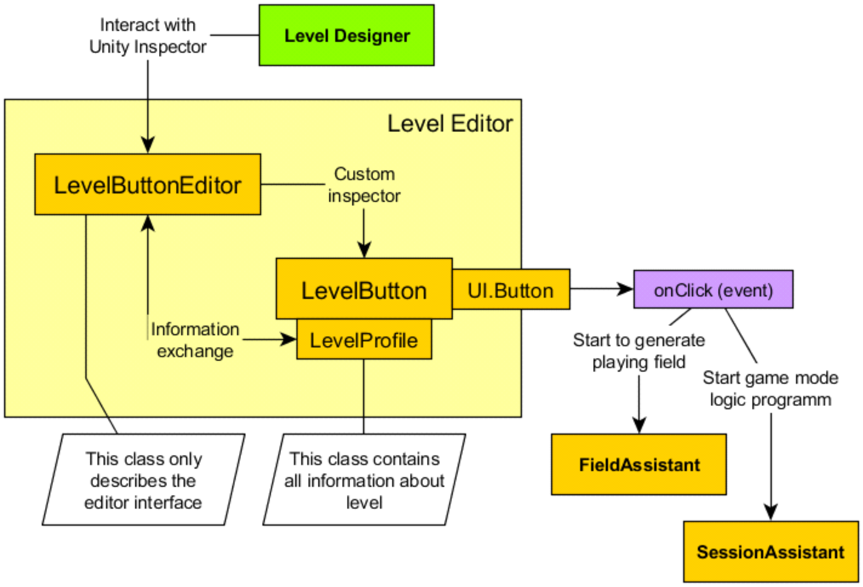
The documentation of **Soomla Store** plugin contains the information about adjusting the publisher account.

Soomla Store integration:
http://know.soom.la/unity/store/store_gettingstarted/

LEVEL EDITOR - THE WORKING PRINCIPLES

Level editor is made using Custom Inspector for **LevelButton** component.

LevelButton - component of level starting button. It contains **LevelProfile** type parameter, that contains all necessary information about level. Using this parameter level editor change the arrays into understandable form.


Interact with Unity Inspector
Level Designer
Level Editor
LevelButtonEditor
Custom inspector
LevelButton
UI.Button
onClick (event)
Information exchange
LevelProfile
Start to generate playing field
Start game mode logic programm
This class only describes the editor interface
This class contains all information about level
FieldAssistant
SessionAssistant

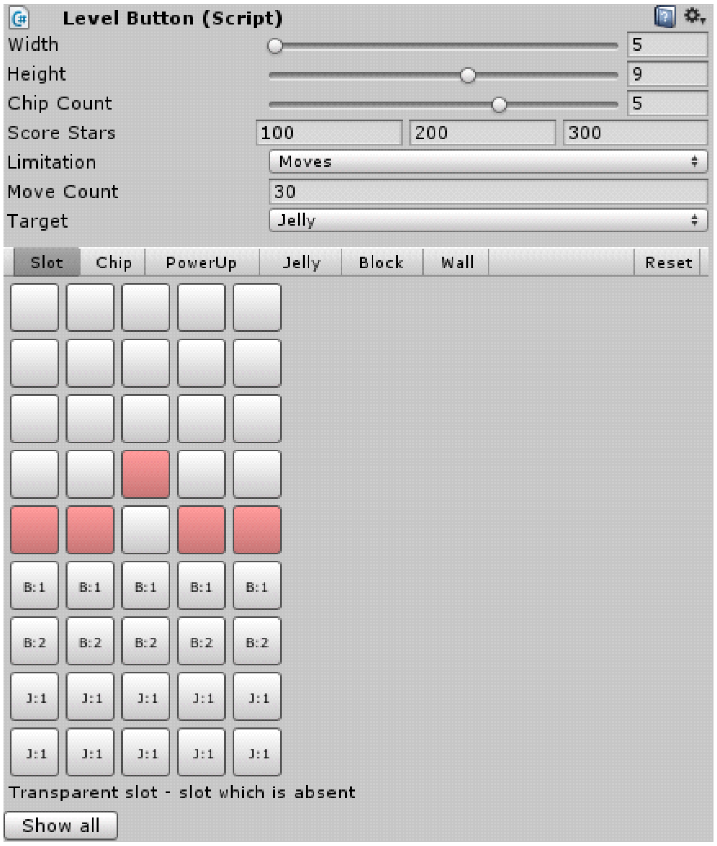
HOW TO CREATE A NEW LEVEL

To create a new level you only need to create new button UI.Button and add the **LevelButton** component. However, if you work in demo-scene then for creating the new level you only have to duplicate one of the available levels.

In demo-scene you can find the levels here: *«UI / LevelList / ScrollRect / Grid / … »*. The new duplicate of level would have the design inherited of original level. To clear the level you should press the *Reset* button.

LEVEL EDITOR - INTERFACE

Level editor looks like:



Level Button (Script)

Width 5

Height 9

Chip Count 5

Score Stars 100 200 300

Limitation Moves

Move Count 30

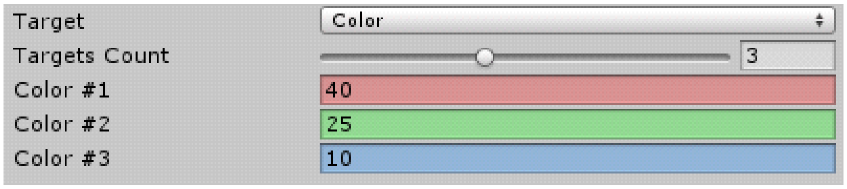
Target Jelly

Transparent slot - slot which is absent

On the top you can see the panel with level settings and game modes:

- *Width* and *Height* parameters – the size of game field.
- *Chip Count* parameter – the count of chip's colors.
- *Score Stars* – the count of the score points necessary to getting the first, the second and the third stars.
- *Limitation* parameter – limitation type on level. You can choose to limit the player by time or by count of moves.
- *Move Count / Duration* parameter – the size of limitation depending on chosen limitation mode. In the case of limited time, parameter is given on seconds.
- *Target* parameter – target parameter which need to achieved for level completion. There are also options of targets:
    - *Score* – get the count of score points. The count of score points equals to count of necessary to get the first star.
    - *Jelly* – destroy all jelly on the level. The existence of jelly is necessary to avoid the level finished immediately.
    - *Block* – destroy all blocks on the level. The existence of blocks is necessary to avoid the level finished immediately.
    - *Color* – destroy the certain count of chips of certain color.

In the case if Color mode is chosen there are many parameters:

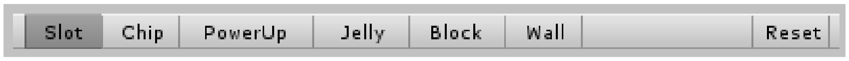


Target	Color
Targets Count	3
Color #1	40
Color #2	25
Color #3	10

- *Targets Count* – the count of color targets. How many chips should player to collect.

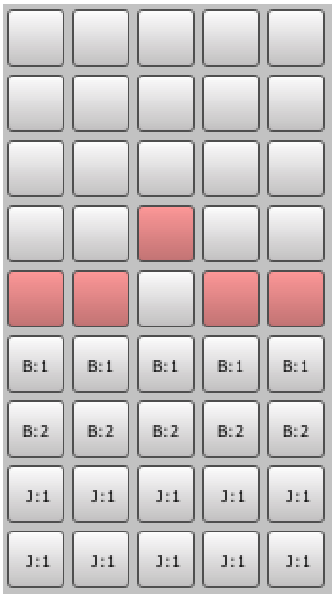
- *Color #N* parameter - the count of certain color chips that player should collect. (Colors are conditional - while level generation they could be changed on others)

Below is instrument panel. Here you can choose one of the field edit modes.



- In *Slot* mode you can choose which slots would be absent on the level. It is necessary to give the level more complex forms.
- In *Chip* mode you can choose what color of chip would be in some slot. There are some options - if slot become grey it means that in generation moment chip would not be in this slot. If slot is white it means that in generation moment there would be chip in this slot and chip would have a random color. You can give a slot one of the conditional colors. Slots with the same colors would take the chips with same colors while generation. But you should keep in mind that colors are conditional. You can only adjust the colors mask, which means that on the generated level colors would differ, but pattern would remain.
- In *PowerUp* mode you can put in slot one of the three bombs. If slot does not have the mark, it means that it would not contain the chip or it would, but only simple one. XB - CrossBomb, B - SimpleBomb, CB - ColorBomb.
- In *Jelly* mode you can adjust the place where jelly would locate. Jelly could have many level - three at max.
- In *Block* mode you can adjust the place where blocks would locate. Blocks also could have many levels - three at max.
- *Wall* mode allow to set the walls between slots, that would impede the movement of chips between this two slots.

Below you can see the pre-view of the game field. The pressure on some slot changes its properties depending on chosen editing mode.



		🟥		
🟥	🟥		🟥	🟥
B:1	B:1	B:1	B:1	B:1
B:2	B:2	B:2	B:2	B:2
J:1	J:1	J:1	J:1	J:1
J:1	J:1	J:1	J:1	J:1