28/07/2023

Blocky VR - Documentation

Killian PERRIN, Lucas BOLBÈNES <lucasbolbenespro@gmail.com>

You can comment and ask questions about the original gdoc at this address :
https://docs.google.com/document/d/1iNEQIweqEoIrNt98pjoVjeJm7u5pIhaiYaA5h_mLo58/edit

Introduction

User Manual

1. Moving

Point to the place on the floor where you want to go and press the grip button. To turn around, use the left joystick.

2. Placing a Block

You can use either the left or the right controller to hold a block. Keep the grip button pressed when you want to hold the block and release the button when you want to release the block. To place the block, the user has to target the right socket following the common rules of coding.

3. Running the Program

When you want to run the code, simply press the trigger button while targeting the EXECUTE button. If desired, you can press the pause button to modify a part of the code that has not been executed yet.

4. The Variable Block and the Constant Block

To choose the variable's name or the constant's value, you have to use the right controller. Point the right controller at the variable block or the constant block, and then use the joystick to select the name or the value. Once your choice is made, grab the cube using the grip button. After holding the cube, it will not be possible to change it.

5. Deleting a Block

Point the right controller at the block you want to delete. Then press the A button to display the delete window. To confirm the deletion, keep the A button pressed, point the delete button with the right or left controller, and press the trigger button.

Brief presentation of the game

The aim of this project is to create a Blocky-type game in VR. The game contains a library of blocks, a `MainBlock` with an `Execute` and `Pause` button, a randomly generated level with a character and obstacles, and a finish represented by a flag. The player must be able to grab the different blocks and place them after the main block. A chain is formed. When the `Execute` button is pressed, the algorithm represented by the chain is executed and the character performs the actions indicated.

Introductory information

- Please note that there is a misnomer about blocks. There is a difference between scripts and prefabs, but in the following we will talk about blocks in both cases. Rely on the context of the explanation.

Conceptualisation

The `GameManager` class

The `GameManager` class contains references to a number of important objects in the game. These include :

- The character : `character`
- The controllers :
    - `principalHandController`
    - `secondaryHandController`
- The object in front of the character : `objectInFront`
- A prompt for displaying messages from anywhere in the program : `printScreenTMP`
- And two lists of GameObjects used to reset the level when Execute is pressed :
    - `originalObstacles` : contains objects generated from the start
    - `executionObstacles` : contains dynamically generated objects

The main purpose of this class is to centralise general game information in a single place.

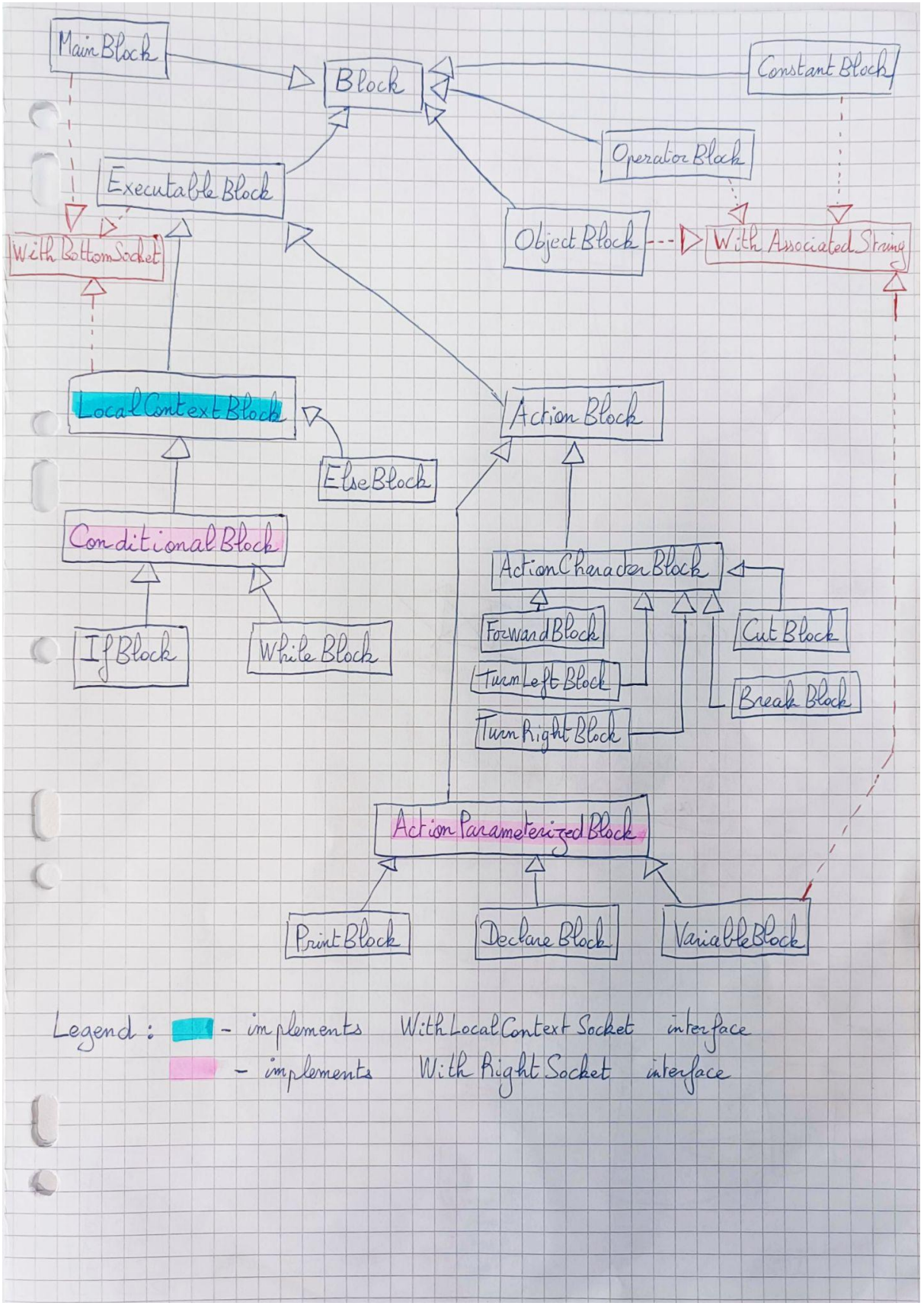
Caution !
To access these different objects, you need to wait until initialisation is complete. This is why we recommend using the `InvokeAfterInit(MonoBehaviour obj, Action action)` method when the action concerns a GameManager object.

`GameManager` also offers some interesting features for programmers:

- Error and warning management: when a user error occurs, it can be reported using one of the following two methods :
    - `ReportError(MonoBehaviour obj, string message)`
      Indicates a critical error to the user by flashing the relevant block in red and displaying a message on the main prompt.
    - `ReportWarning(MonoBehaviour obj, string message)`
      Indicates a minor error to the user by flashing the relevant block in orange and displaying a message on the main prompt.
- Displaying messages on the prompt: `DisplayOnPrompt(string message)`

Script hierarchy for `Block`

Here is a diagram showing the block hierarchy.



- Blocks inheriting from the abstract `ExecutableBlock` class all have a definition for the :
    - `Execute(Dictionary<string, int> variables)`

    - `IsFinished()`.

This is very useful for running the `MainBlock`.

How is the `MainBlock` executed?

The `MainBlock` is executed sequentially. It is important to specify that the `MainBlock` does not contain any list of blocks. Execution simply consists of traversing the chain from socket to socket and calling the `Execute(Dictionary<string, int> variables)` method of each `ExecutableBlock` traversed, then waiting for the latter to finish executing. This is made possible by the `IsFinished()` method.

Local context blocks: `LocalScopeBlock`

These blocks have the particularity of having an additional socket compared with conventional blocks. We'll call it the `localContextSocket`. If a block is in the `localContextSocket` of another block, then it is its referent block.

Variable management

Use of a `Dictionary<string, int> variables` attribute created in the `MainBlock`. This contains all the variables and the reference is passed to the `ExecutableBlock` by calling the `Execute(Dictionary<string, int> variables)` method.

About `Character` prefab

The character prefab contains various child :

- `Bear`, which corresponds to the character's design, and on which the `Animator` is placed to play the animations.
- A `FrontCollider` element containing the `FrontColliderController` script and a `Sphere Collider` placed at the front. These components are used to detect objects in front of the character and update the `objectInFront` attribute in the `GameManager`.

Complicated aspects

Indentation management in `LocalContextBlock`

The aim is to know the number of blocks within a `LocalContextBlock` in order to correctly translate its `bottomSocket`. To do this, we need to transmit information between the blocks.

Here's how it works:

- Each block has a value whose default value is 1.
- The weight of blocks of type `LocalContextBlock` is the sum of the blocks contained after their `localContextSocket` + 1.
- When a first block is added to the `localContextSocket`, it is initialised with the `LocalContextBlock` as the `referentBlock`, then a route is taken through the several `bottomSocket`. If the block added has no next, nothing happens, otherwise each block takes the `LocalContextBlock` as its `referentBlock`.
- The same principle applies when the first block is removed. For this to work, a reference to the first block is kept.
- When a block is added to or removed from the chain, similar behaviour occurs.
- The elements managing this system are :
    - Within the `Block` class :
        - The `value` attribute
        - `InformReferent()`
        - `TransmitNextBlocksToReferent()`
        - `ResetNextBlocks()`
    - Within the `LocalContextBlock` class
        - `BrowseChildAndUpdate()`
        - `TranslateBottomSocket()`
        - `ResetAllBlocks()`
        - `TransmitAllBlocksToReferent()`

Role of the EditBlockInSocket script

This script is useful in a specific context. When a block B is added to the socket of a block A, the `EditBlockInSocket` script placed on block A can then control the bottom and right sockets of block B. It can either deactivate them or modify their layers. This is very useful when adding a `VariableBlock` block to the condition of an `IfBlock`, because in this case the `bottomSocket` of the `VariableBlock` must be deactivated and the layers of its `rightSocket` must be modified.

Conclusion

Possible improvements

- Add boolean constant blocks (true and false).
- Remove the shelves controllers that use the joystick to change value, they all work in the same way but they currently have separate scripts. This could be generalised.
- Add the ability to create functions, as the blockchain can quickly become too long.

A final word

Unfortunately, this documentation is incomplete. Please don't hesitate to contact Lucas Bolbènes<lucasbolbenespro@gmail.com> for further information.
You can also comment and ask questions about the original gdoc at this address : https://docs.google.com/document/d/1iNEQIwegEoIrNt98pjoVjeJm7u5pIhaiYaA5h_mLo58/edit