Technical Documentation of SPK Game

Intro

This documentation gives you a general overview of the overall architecture of the game in unity and some additional information about the “game manager scripts”, which handle the “state” of the game, in and in between levels.

All other documentation is done in code and unity itself. It makes no sense to write it down here, because its always needs the context of a certain game-object and other connected items. We gave our best to:

- name methods and variable appropriately, so that they describe what they stand for or what they do
- structure files and folders so, to find what you look for quickly
- write tooltips which show up when you hover over an item in Unity Inspector

Manager Scripts

Manager Scripts are named “Master” followed by the type of Level, i.e. ***MasterPuzzleManager***. These Manager-Scripts are connected to the MainCamera in each scene.

These scripts manage game state for each level and in between levels. You find the definition of Level-States in Scripts/Managers/GameStates.cs

Since all Levels depend on other Levels, e.g. when entering the book from Adventure Level, we need to know if the Puzzle is solved in order to decide if we should load puzzle again (unsolved) or to load action level.

Manager Scripts subscribe to and broadcast / fire events based on a Event-System using ***Messenger***. Messages are mainly used to talk to User Interface. For instance: we show a user-interface element when a puzzle is solved. Or we show an other UI-Element when a user enters a level for the first time, the Puzzle-Manager-Scripts fires an event called “FIRST_TIME_IN_PUZZLE”

User Interface Manager UI-Managers

UI-Manager Scripts are similar to Manager Scripts. They register listener like “SHOW_STARTUP_PLATE” and, when this event is triggered they show the corresponding UI-Element.

Master Scene and debugging single levels

When you start the MasterScene you can set the order of the levels. There, on Game-Object UI_PERSISTENT the MasterGameManager script lives. MasterGameManager Scripts is a single instance (singleton), which the other Manager Scripts are using to ask for the state of other levels. It has only two relevant methods: a) saveSceneState b) getSceneState

When you want to work on a single level, you can load it directly in Unity. Anyway, since it depends on the overall state of the game, this state is not given when level is started directly. Therefore you can manually set state in the file/script Managers/GameStates. So if you want to debug the puzzle, but it always says: “I am missing the things you need to collect first”, you can set in Managers/GameStates that these are already colllected. You would do this in the state of the preceding Adventure.

Abbreviations

GO = Gameobject

top

Adventure Part

This is an overview of the most imporant Scripts and GO in Adventure levels.

Scripts:

MasterAdventureFlex (or similar)

on GO MainCamera

CharacterMovementMouseFlex

on GO Player Here you can setup certain things, like the layers of the the Collectibles, Enemies, etc. Gridfactor: defines the internal grid. The size of “Floor” is divided by gridfactor. Cam Close / Far : Zoom-Factor of cam (when pressing SPACE)

UiManager_Adv_01 (or similar)

On GO UI

Gameobjects

Floor / Astar

The definition of walkable and unwalkable areas is done here, via the plugin “A* Pathfinding Project”. The Scale of Floor is important

Collectibles_DONOTRENAME

The parent GO of the items a user have to collect. Each GO in here can be collected (add a collider). The name of these GO are important. They need to be set up in the following puzzle level. Set the names in Puzzle Manager (there is a field where the name must be entered)

top

Puzzle Part

The Puzzles (generally) consist of

1. movable parts (placed in upper left corner)
2. a kind of skeleton, or a (faked) kinematic chain
3. some trigger areas which, when triggered by a) activated b)

Example Puzzle_01

The Pin in PuzzleParts/Draggables has a script called *ToggleLookUp*. Here we set a variable (other) to the GO of *CenterRotation* and an Activation Zone (a collider2D), which enables OnTriggerEnter the Script *LookAt* in other. So basically the whole kinematic chain consist of these two scripts *LookAt* and *ToggleLookUp*

top

Action Part

This is an overview of the most imporant Scripts and GO in Action levels.

Scripts:

MasterActionManager

on GO MainCamera handles “states” and talks to other levels

SlotsController_A19_multiple (or similar)

on Slots handles the Animation of Slots and Gameobjects in Slots

UiManager_Action

on Canvas Script is subscribing to Game-Events (like when user catches a thing via the laser beam) and shows so called UI-Plates based on them. It is also do the counting of how many tries etc.

UICatchedUpdate_A19_multiple

on Canvas/ObjectsToCatch Here you set the “What to catch UI” in UI. This defines the length of a gameplay round. You need to make sure that the names do match the once in Slots “What to catch” field (SlotsController_A19_multiple) Also you can set Materials here for Active, Default, Inactive

CatchSlots_A19_multiple and PointViaMouse_A19

on Arm (or other mouse following GOs) these two scripts allow you to define how the “laser” and “catch mechanism” is working

Gameobjects

Slots

The Child GO of “Slots” can be moved around, duplicated or deleted. Each Slot contains a “Hole” which must be named “Hole_DONTRENAME”. The Script *SlotsController_A19_multiple* takes its GOs and appends it to one of the slots The Transparency of the holes is done via a shader and is only working on certain other shader (See the shader, stencil shader of “marvelous techniques”) the holes also have an Animator and an other, simple Script called “SlotAnimationEventHandlerSPK”

Arm (or other mouse following GOs)

the childs (end, start) define the direction of the laser

Animation of the “Slots”

The Animator has to States, starting with default “pop up” slot animation. When “Shuffle Time” in Script “SlotsController_A19_multiple” is over, this calls a function (closeSlot in SlotAnimationEventHandlerSPK) and triggers the closing animation. When the Gameobjects displayed in the Slots gets animated, than make sure, that this animation fits into “Shuffle Time”.

top

More Info about how parts get connected

There are 2 general scripts you find in every level:

1. On MainCamera there is a script called “Master XXX Manager” (or similar). I call this LM (Level-Master)
2. On the top-level UI GameObjects, usually named “UI” or “Canvas” there is one called “UI XXX Manager” (called UIM)

These two general scrips handle:

1. manages the ‘state’ of the level, intra and inter Level based. So the Master Puzzle Manager takes care of “state”-variables like “is_solved” or “has_visited”.
2. usually the UI-Manager subscribes to some Events (like “FIRST_TIME_IN_PUZZLE”) and run functionality based on them. In case of FIRST_TIME_IN_PUZZLE it is showing an UI-Plate with first time instructions

The Master Manager (LM) is doing one more thing: it is reporting its own state to the (real) master-script in scene Master (called GM (Game Master). Each time the level gets loaded, this script asks GM for a former state (like in Adventure to see if the user visited the book already or solved the puzzle). If there is a former state, it takes it and sets its own (state) variables. It also starts a Coroutine which reports the state to GM in a fixed period (this is a bad design pattern, but I found no other way yet, since on “OnDestroy” (called by

Unity when level unloads) some important info is not available anymore (cannot access any gameobject anymore)).

Example Shared Data

The Adventur part needs to know about the state of Puzzle and Action Parts. For instance, when user visited the book, the items to collect will be displayed. Or when the puzzle is solved and the book is visited again, Action part shall be displayed.

Level - Manager

Defines the connected levels (Adventure - Puzzle - Action) Has some type specific settings but the general purpose and architecture is the same. Method *writeState* reports state to GM Method *retrieveState* retrieves state from GM

Mock Data

when you work on a single level, and therefore there is no *master*-scene, you can provide so called mock-data in LM. In this mock-data (which is currently commented out) you can set the *state* manually and do some testing or further development of a single level, without the need to run the master each time.

Serialized Data

At the top of each LM you find a small class which defines the interlevel *shared-state-data*.