**Quick Start Guide: Basic Design and Development Principles**

*Build 220724*

0. Foreword

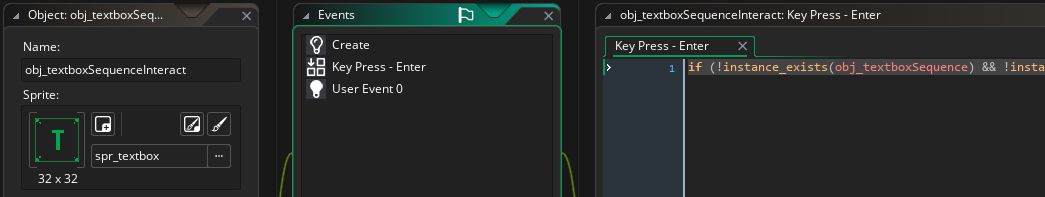
When developing a game with The OQRP Engine, there are several developmental guidelines that deviate from standard Game Maker 2 creation practice. This guide will show you the most basic and fundamental principles to help get you started on your project. Keep in mind that you need to follow these guidelines **strictly,** and **throughout the entire development of your game.**

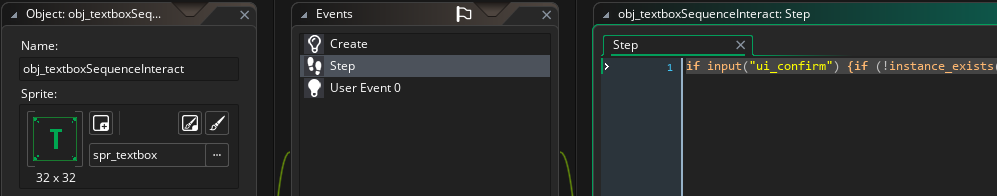
1. Using Controls in your Game

The OQRP Engine overrides the default Game Maker control scheme and adds additional functionality that is not normally supported. When implementing controls in your game, there are a few things you need to take into consideration.

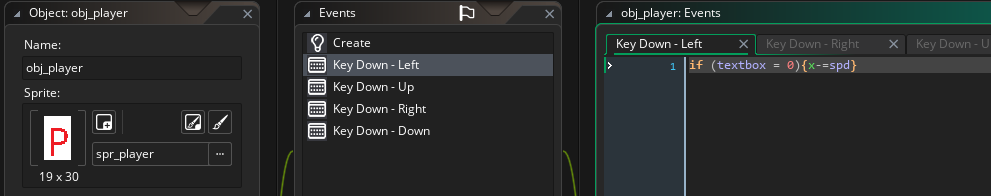
**1a. Events and Checks**

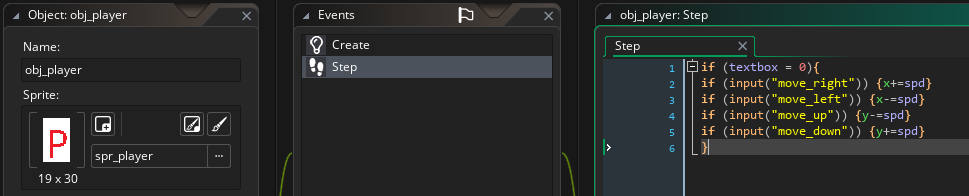
Instead of using Game Maker’s default control events, use the function input(action\_id) inside your regular events (most commonly Step, Begin Step and End Step but you can also use others for single-frame checking). These functions return a boolean value. Here are some examples:

**Textbox Interaction** – Here’s how you would do it in regular GMS2:

and here’s how to do it in the OQRP Engine:

In this case, if input(action\_id) completely replaces the key press event.

**Player Movement** – Here is a basic player character:

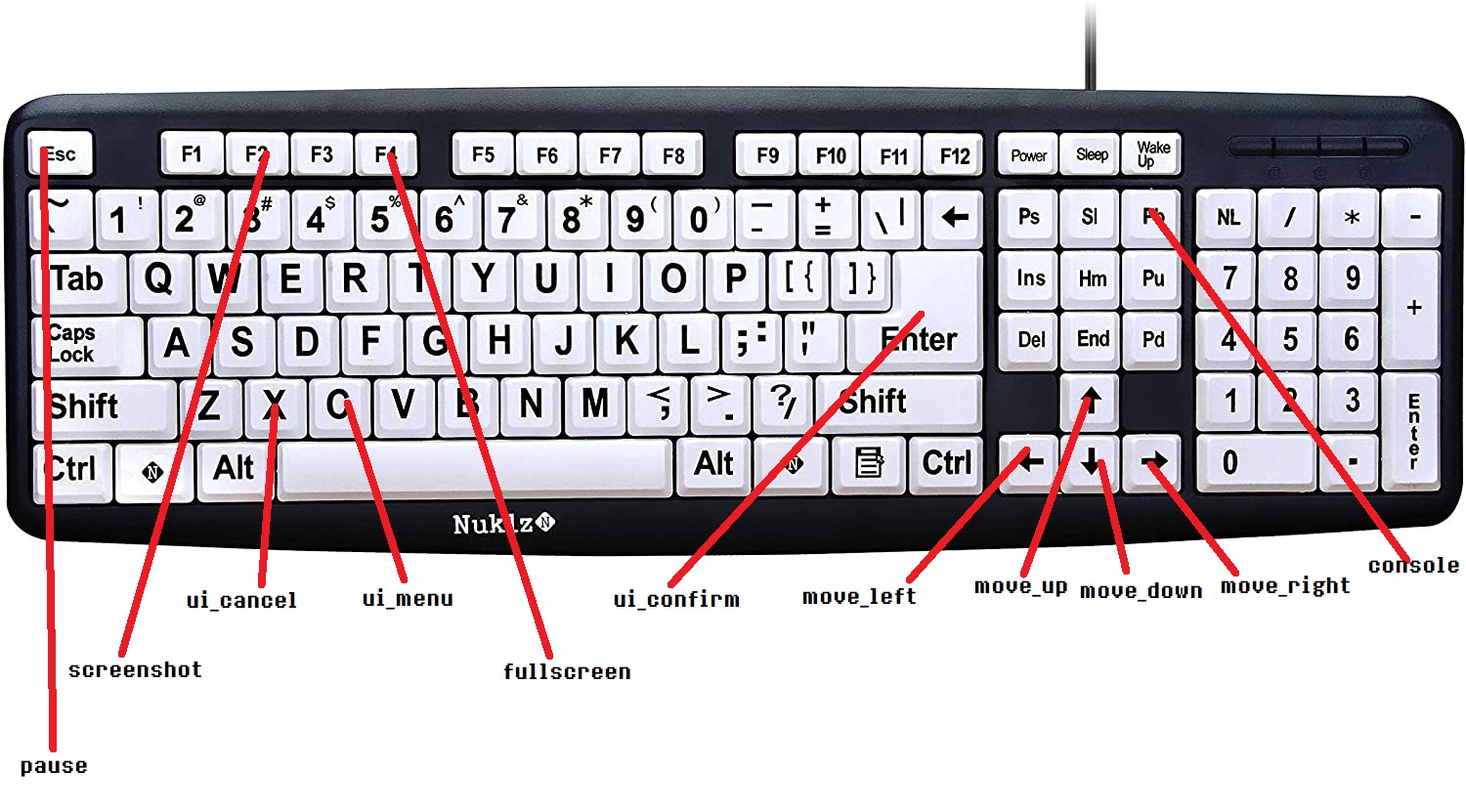
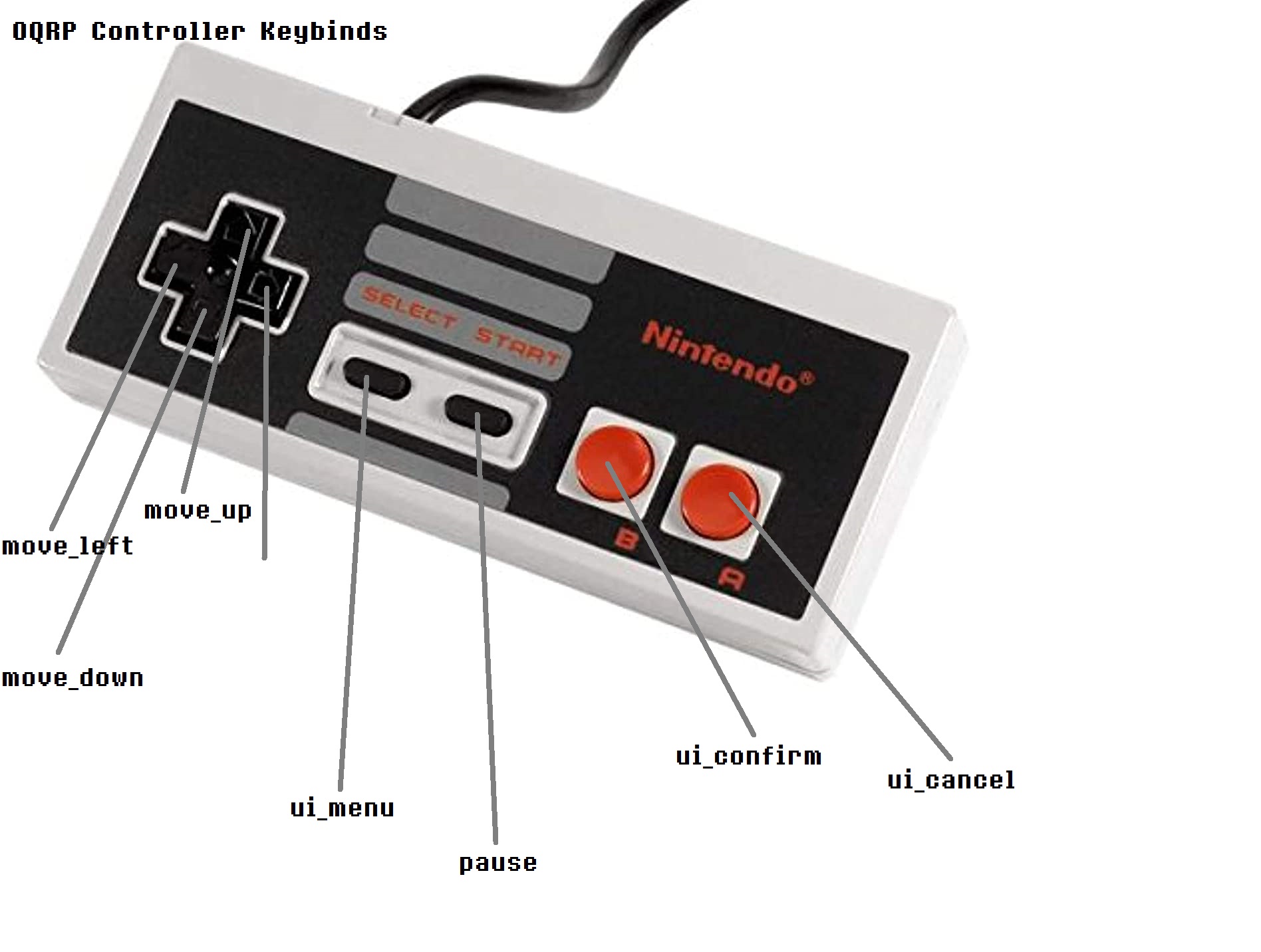
and here’s the OQRP implementation:

Once again, if input(action\_id) completely replaces the key down event.

While this may make your code messier, it allows you to effortlessly use custom dynamic single- or multi-keybinds, use other input devices like controllers and switch between them „on the fly“. This is not possible with Game Maker’s standard key events.

**1b. Action IDs**

The OQRP Engine uses Action IDs to determine player interaction regardless of platform or input device. Below are images of default control schemes and mappings, using a PC keyboard and NES controller as references.



2. The Saving/Loading System

The OQRP Engine uses the .OQRPSAV file format to store saved games. This is an encrypted and signed file format with the purpose of detecting and protecting against possible safe file corruption and modification, i.e. cheats. This approach has a few downsides. To prevent the player from downloading someone else’s completed game, the save files are tied to the player’s device. This may unfortunately break or invalidate a player’s save files in a number of scenarios, including a hardware change or game reinstall. The system is slot-based; the developer can choose how many save slots their game can use.

**2a. Initialization**

Upon startup, the system detects a first launch scenario and initializes the slots. Alternatively, the system loads information about existing save slots.

**2b. Saving**

Everything that needs to be included in a save file has to be stored in the global.save ds\_map. It is entirely up to the developer to create their own ds\_map save structure. To make manipulation of this map easier, the functions varSave(name, value) and varLoad(name) can be used. To push a save file from memory to the disk, run the function Savefile().

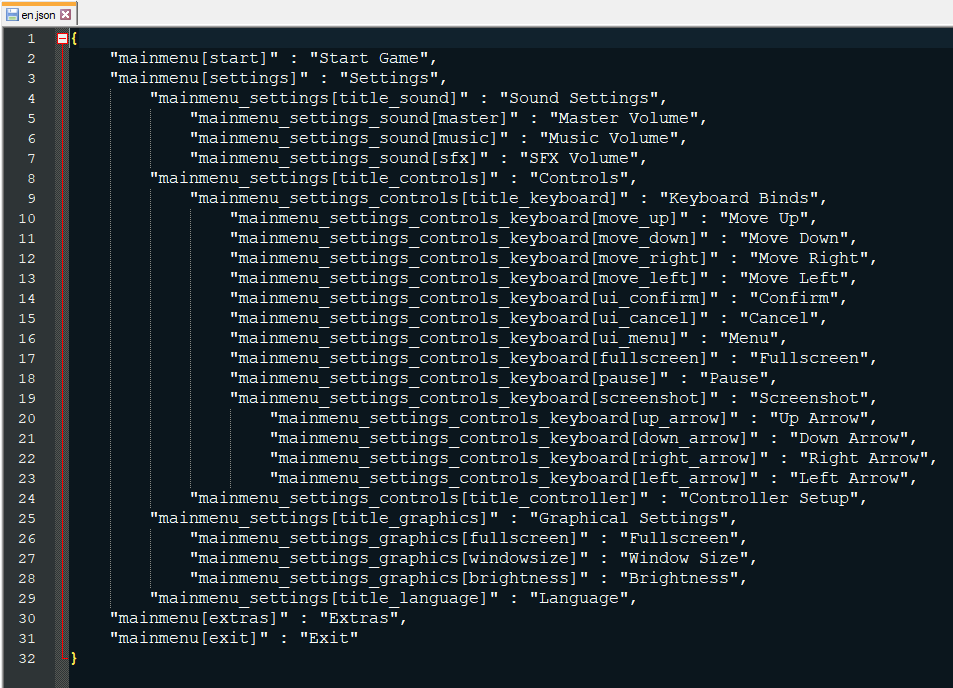
**2c. Loading**

The function Loadgame(slot) loads the slot with the provided ID to memory and checks its integrity. If the save file is corrupted, the function forces an exception.

3. Written Text visible to the Player

One of the design goals for The OQRP Engine is an advanced localization system that supports switching between many different languages „on the fly“, affecting not just the text in-game, but also every aspect of the UI.

**3a. Understanding the script file**

Below is a sample of an english oqrp script file:

Language files are stored in the datafiles\lang\ directory. Notice that this file is stored in a JSON format, as it needs to be human-readable and human-editable, so that translators can quickly and easily make changes to the file. The easiest method for editing this file is to use a JSON editor, but it can also be edited by hand using any available text editor.

Keep in mind as this is very important: **Every line of text that the player can see in the game, including UI, textboxes, descriptions, items, titles, etc... needs to be stored in this file and acessed through the code! Do not insert player-readable strings into the game’s code! Refer to section 3b on how to use this file!**

**3b. Reading the Script File**

To acess contents of the script file, use the function s(key), where key is the unique identifier given to the string, i.e. „mainmenu[start]“. This function returns the requested human-readable string. Here is an example of how to use this function:

Drawing text to the screen:

draw\_text(0, 0, „Hello World!“) – incorrect

draw\_text(0, 0, s(„text[helloworld]“)) – correct

and here is how the text[helloworld] key looks like in the script file:

„text[helloworld]“ : „Hello World!“, 🡨 don’t forget the comma at the end (if there is another entry below it)

The obvious advantage of this system is that the key for the given string stays the same, regardless of selected language. Changing languages can therefore be done instantly, and no actual changes to the code have to be done. It also allows translators to modify the script file without needing to modify or even see the code. Translations to multiple languages can be done simultaneously.