

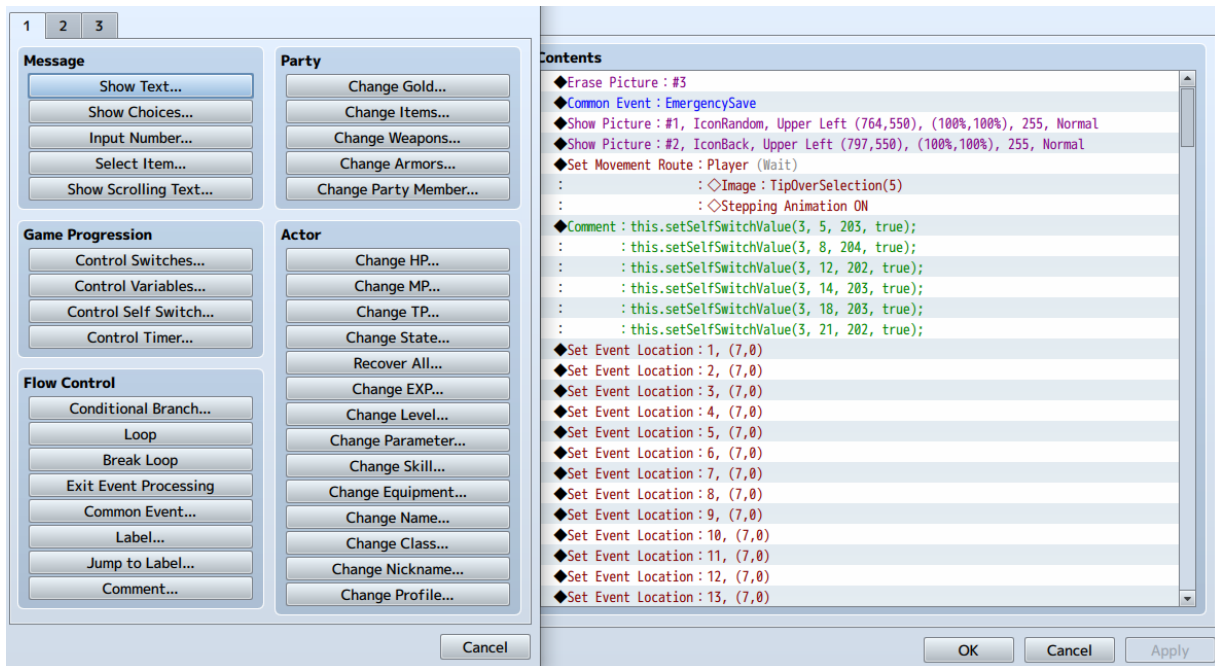
This document will cover my TipOver showcase game. If you haven't already, I recommend playing it first for a few minutes before continuing as it will give you a better understanding of what you're reading for the next 20 pages.

## The Basics

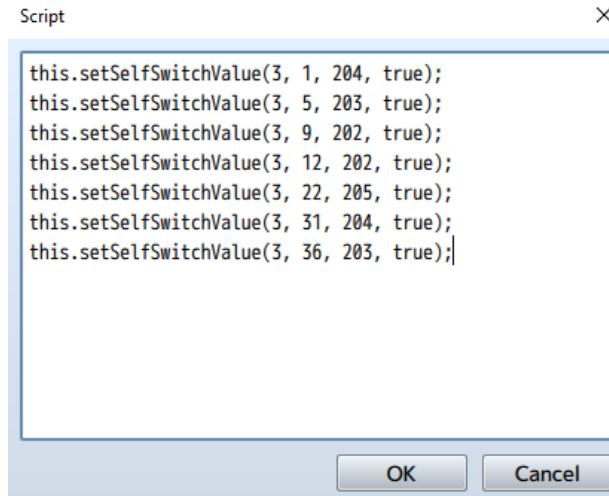
### What is RPG Maker MV?

The RPG Maker program is a series of game development tools that are used by developers to bring their game ideas to life without having to start from scratch. RPG Maker MV was released in 2015 and runs off of JavaScript, so that published games have the option of running on mobile devices as well as PC executables.

Most of the programming that occurs in RPG Maker is drag-and-drop, so that users with even only the most basic understanding of programming can still create a wide variety of program methods ranging from dialog trees to logic puzzles. However, the premade code options do not cover everything that a developer may wish to implement, and thus there is the option to write and run custom JavaScript code.

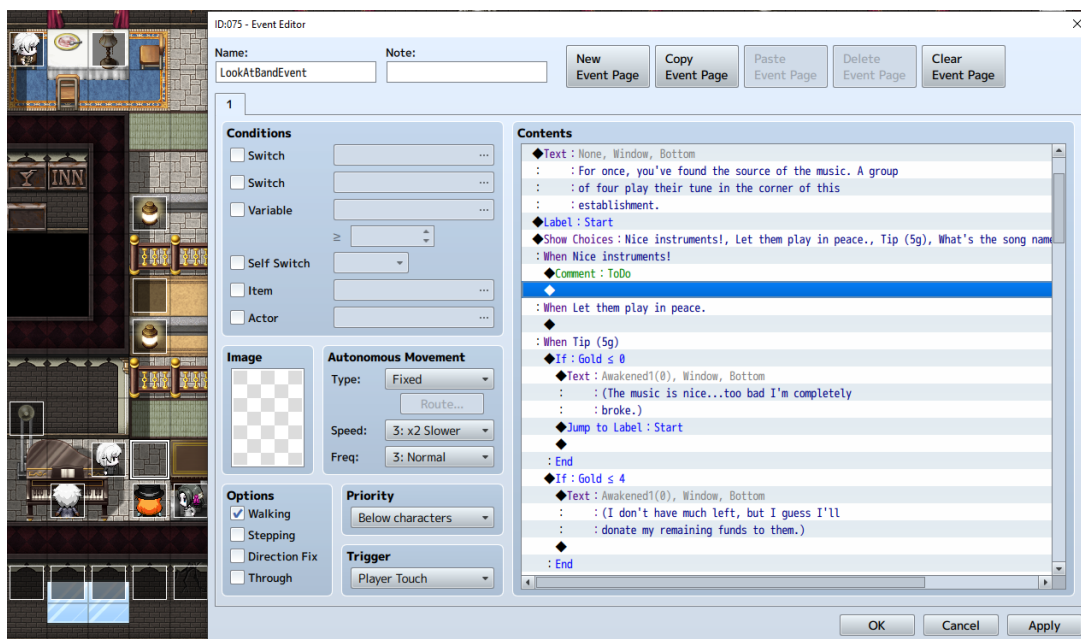


*Code written using the available drag-and-drop methods.*

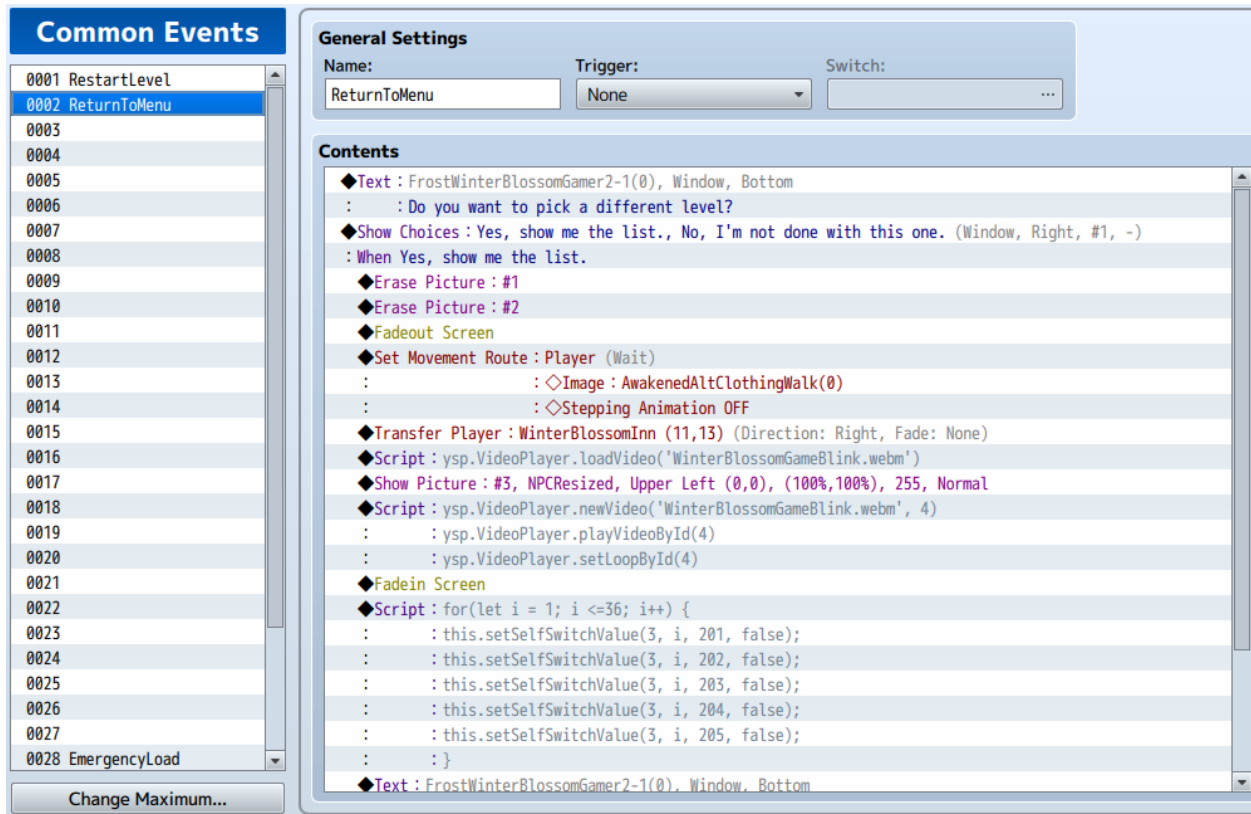


### *Custom JavaScript commands.*

One more concept to note about RPG Maker is that almost all user code runs in containers called Events. Events can be local to individual Maps (which are the areas that a player interacts with in a game), or they can be global to all Maps (called Common Events). For example, an NPC (non-player-character) entity that a player can talk to in a Map is just an Event with an image of a person on it. When the player attempts to talk to the NPC, they activate the code within the NPC's Event which can lead to dialog, variable adjustments, or anything that the developer wishes to occur. Any Event is able to call upon a global Common Event, which is often a preferred method to avoid repeating code in multiple Map-local Events.



*An example of a Map-local Event which triggers a dialog tree when the player characters steps onto the Event's location in the map.*



*An example of a common Event which is not tied to any specific map. Common Events have to be called by other Events, or can be triggered automatically based on the value of a variable.*

## What is TipOver?

TipOver, originally *The Kung Fu Packing Crate Maze* by James W. Stephens (<http://www.puzzlebeast.com/crate/index.html>), is a puzzle board game produced by ThinkFun. The rules are fairly simple: given a prearrangement of the game pieces within the 6-by-6 game board, the player must navigate their own game piece to the end goal piece by tipping over towers to create paths. The narrow size of the game board as well as the inability to tip towers onto blocking obstacles can create varying levels of difficulty. The ThinkFun version came with 40 levels that were rated from Beginner-level difficulty to Expert (which I ‘borrowed’ for my digital version). While rated ages 8 and up, a few of the harder levels can certainly give even an adult cause to pause and think. While a digital version is nothing new with a Flash game existing for the 5-by-5 variant of the game, I am probably the first person to recreate the game in RPG Maker.



*The physical game (original image belongs to ThinkFun).*

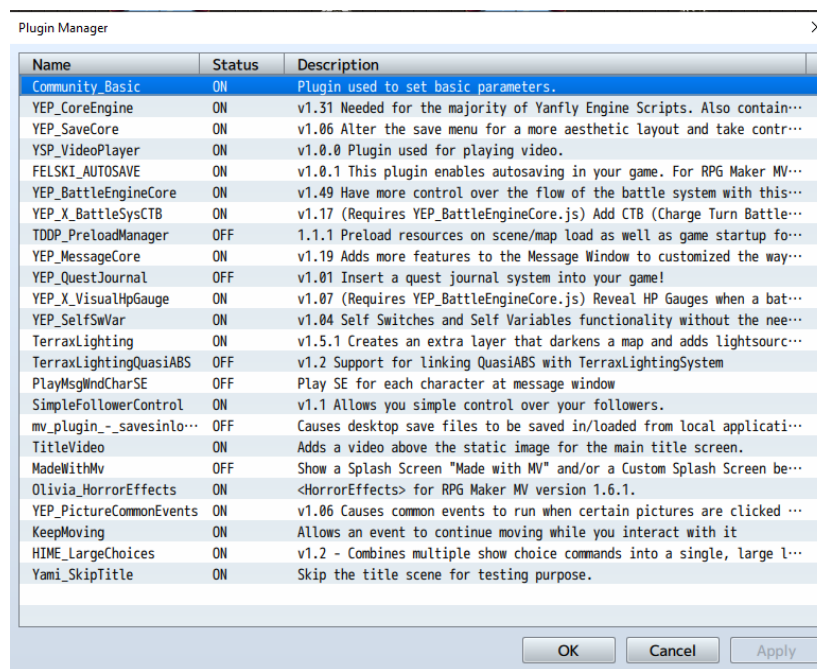


*The digital version that I created in RPG Maker MV.*

# How It Works

## First, the plugins:

Most technically-inclined people know what a plugin is: It's a bundle of code that can be loaded into a larger program to enhance its functionality. Anyone who has an ad blocker or password manager for their browser is using a plugin. For RPG Maker MV, there exists a variety of plugins that do all sorts of things from enhancing the visual looks of the menus to adding quest journals and much, much more. Some of these plugins are free, and some of them cost real money that developers are willing to pay simply because of how much useful functionality the plugins can bring to a project.



Name	Status	Description
Community_Basic	ON	Plugin used to set basic parameters.
YEP_CoreEngine	ON	v1.31 Needed for the majority of Yanfly Engine Scripts. Also contain...
YEP_SaveCore	ON	v1.06 Alter the save menu for a more aesthetic layout and take contr...
YSP_VideoPlayer	ON	v1.0.0 Plugin used for playing video.
FELSKI_AUTOSAVE	ON	v1.0.1 This plugin enables autosaving in your game. For RPG Maker MV...
YEP_BattleEngineCore	ON	v1.49 Have more control over the flow of the battle system with this...
YEP_X_BattleSysCTB	ON	v1.17 (Requires YEP_BattleEngineCore.js) Add CTB (Charge Turn Battle...
TDDP_PreloadManager	OFF	1.1.1 Preload resources on scene/map load as well as game startup fo...
YEP_MessageCore	ON	v1.19 Adds more features to the Message Window to customized the way...
YEP_QuestJournal	OFF	v1.01 Insert a quest journal system into your game!
YEP_X_VisualHpGauge	ON	v1.07 (Requires YEP_BattleEngineCore.js) Reveal HP Gauges when a bat...
YEP_SelfSwVar	ON	v1.04 Self Switches and Self Variables functionality without the nee...
TerraxLighting	ON	v1.5.1 Creates an extra layer that darkens a map and adds lightsourc...
TerraxLightingQuasiABS	OFF	v1.2 Support for linking QuasiABS with TerraxLightingSystem
PlayMsgWndCharSE	OFF	Play SE for each character at message window
SimpleFollowerControl	ON	v1.1 Allows you simple control over your followers.
mv_plugin_-_savesinlo...	OFF	Causes desktop save files to be saved in/loaded from local applicati...
TitleVideo	ON	Adds a video above the static image for the main title screen.
MadeWithMv	OFF	Show a Splash Screen "Made with MV" and/or a Custom Splash Screen be...
Olivia_HorrorEffects	ON	<HorrorEffects> for RPG Maker MV version 1.6.1.
YEP_PictureCommonEvents	ON	v1.06 Causes common events to run when certain pictures are clicked ...
KeepMoving	ON	Allows an event to continue moving while you interact with it
HIME_LargeChoices	ON	v1.2 - Combines multiple show choice commands into a single, large l...
Yami_SkipTitle	ON	Skip the title scene for testing purpose.

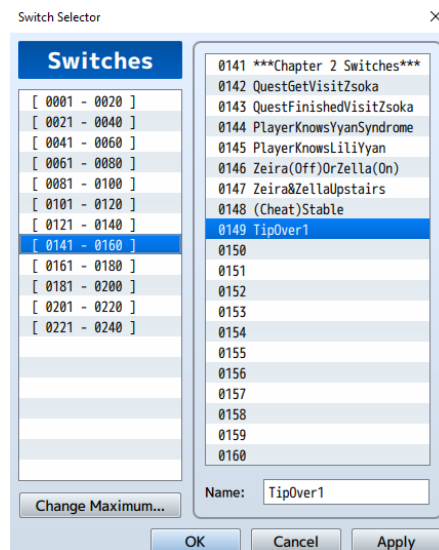
*The Plugin Manager window for RPG Maker MV.*

Don't be daunted by the above list of plugins, some of them aren't even used in this particular project as they're leftovers from my main game which I pulled this smaller demo project from. As for the TipOver game itself, we're only concerned with one plugin: *YEP Self Switches and Variables*.

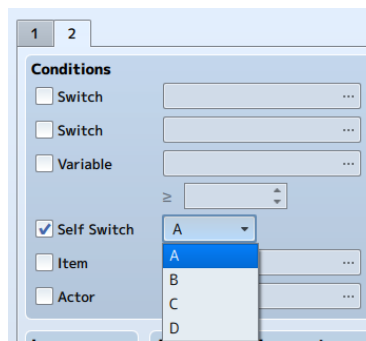
By default, RPG Maker MV has three main options to keep track of variables: variables, switches and self-switches. Variables and switches are global and keep track of Integers and Booleans respectively (variables can also hold other data types, but that will not be relevant for this project). Self-switches are similar to regular switches, but they're local to their host Event and don't take up space in the global variable list. Self-switches are useful if you have many small Events that alternate between two or more states and you don't want to manually assign

different global switches for each state in each Event. For example, if the game has 100 treasure chest Events with the two states of being ‘locked’ and ‘unlocked’, you can tell each Event to look at the state of one of its own self-switches to determine what state it should be in. When the player character unlocks one of the chests, the particular treasure chest Event will switch its self-switch variable to ‘ON’ without affecting any of the other 99 treasure chest Events.

By default, each Event has four self-switches the player can toggle: A, B, C, and D. In many cases, this is enough. For situations where it’s not enough for internal logic, the developer can use JavaScript to create and toggle more self-switches of their own naming (for example, a self-switch labeled ‘E’ can be created and used as a conditional variable within an Event’s running logic). Keep in mind that the less regular switches that the developer has to assign, the less cluttered their switch list will be in the long run. In my main game project, I have already reached 140 switches in the first chapter alone.



*Although the game allows for thousands of switches and variables to be assigned, the developer is likely to run into sanity issues before they run into data memory issues.*



*Using self-switches whenever possible can help prevent useless switch assignments from cluttering the global switch list.*



However, there is one small issue in the default RPG Maker MV setup, one that can't easily be bypassed by a few lines of JavaScript. An Event will check its assigned conditions before being allowed to exist on the Map. If one or more conditions aren't true, the Event will not manifest to the player. By default for self-switches, an Event can check for self-switches A, B, C, or D. However, an Event cannot use a custom self-switch assigned through Javascript as a condition for existing! Even if the developer writes the script `$gameSelfSwitches.setValue([31, 21, 'E'], true)`; which sets the value of a new self-switch 'E' of Event 21 in Map 31 to 'ON', Event 21 will never be able to use it as a manifesting conditional variable. It can run an internal script to check the value of its self-switches once it's running for internal logic, but as a conditional variable to get the Event itself running in the first place, the developer-assigned self-switch E may as well not exist as the dropdown menu only accounts for A, B, C, and D. It's a catch-22 situation, so to speak.

The brute force alternative is to use regular global switches, but let me show you a preview of the game board in the developer editor first to show you why that solution wouldn't work.



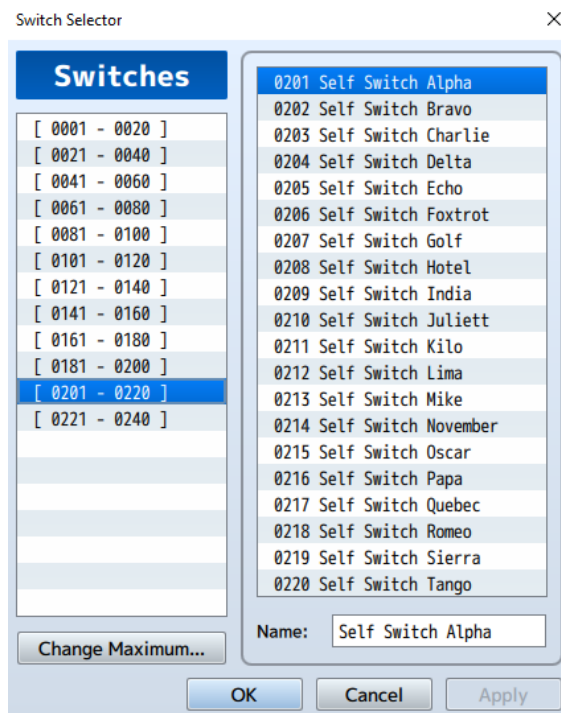
Each of the white squares is an Event. Right now, we're 'only' focusing on the 36 Events that are on the checkered pattern. Each of those Events needs to exist in six different states on command: empty, the goal piece, a fallen stack piece, and the three different types of stack pieces. If I only needed five states from each of the Events, the default A, B, C, D self-switch values would be sufficient, as I could assign A through D to a visible playing piece and have the 'blank' state

trigger when none of the self-switches have the ‘ON’ value (the conditions menu unfortunately only allows for checking one self-switch at a time, so combinational logic would not work).

If I were to substitute in global switches for the Event’s conditional check, not only would I need to add at least 36 additional assigned global switches for this one puzzle, but I would have to come up with the logic for the game to figure out which switch to flip at any point on the board: flipping self-switch A for Event 1 and flipping self-switch A for Event 2 uses the same code but with the Event IDs updated (more on that logic in the next sections). You can’t reuse code for global switches, as each global switch has its own unique id and trying to use just one global switch for all the pieces would cause every spot on the game board to be updated upon the switch being toggled as opposed to just a single game piece. It’s still possible to substitute global switches for a self-switch in this context, but as developers we want to look for the easy way to manage this.

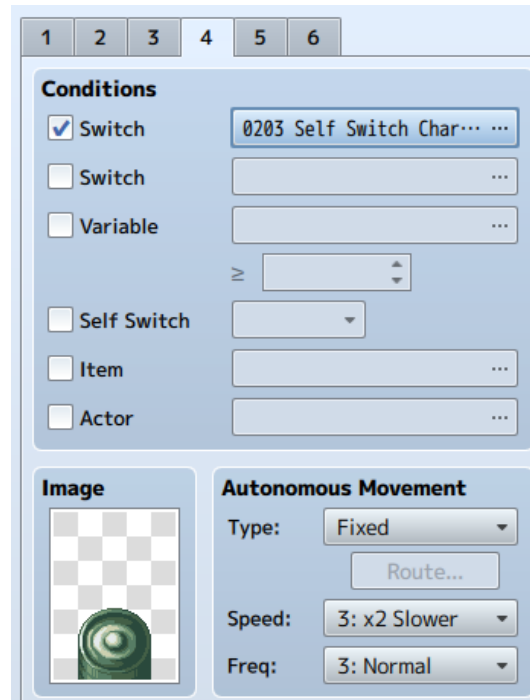
That’s where the *YEP Self Switches and Variables* plugin comes in, and the solution it provides is surprisingly close to the one that I just ranted above on how it would be a bad idea: use switches as self switches.

The way the plugin works is that you allocate the switches that you want to act as self-switches as usual, but with the requirement that they have the wording ‘Self Sw’ or ‘Self Var’ in the name, for switches and variables respectively. I opted to go ahead and allocate the whole NATO alphabet ahead of time for future-proofing, but the TipOver program itself only needed switches Alpha through Echo in this case, so we’ll ignore Foxtrot through Zulu.





In the end, we needed to allocate 5 global switches versus the 36 for the brute-force method. How is this different from normal switches? The plugin ensures that any properly-named switches can be used as self-switches for conditional checking, and that the global functionality of these switches is removed.



*The Event conditional checks treat the special switch as a regular switch, but in reality it has the same functions as a self-switch.*

The best part is that because these special switches affected by the plugin aren't updated globally, they can be reused between events just like normal self-switches. Toggling on switch 0203 'Self Switch Charlie' for Event 4 in Map 3 will *not* actually toggle the value of switch 0203 for any other Events using it, just the one Event that you use the script call for. This lets us get around the A, B, C, D limit of normal self switches for the low price of a small handful of pre-allocated switch spots *that last for the whole development project*, and \$10 (link to the plugin bundle is at the end of this document).

## The Game Logic:



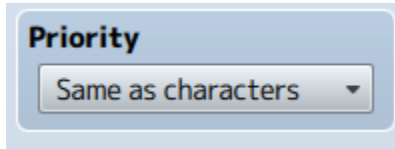
While the plugin in the previous section made some of the back-end programming somewhat easier, it does not, however, program the game for me. I had to do that on my own.

RPG Maker MV provides some default functionality that is useful to emulate this game. For instance, RPG Maker is tile-based, which means setting up a board grid is easy. There is also movement commands for the player character via the keyboard, which aligns with the need to move the player's piece and adjust its facing direction (represented by the white brackets and arrow in the screenshot). Most of the other functionality of the game, however, I had to script myself.

Let's quickly go over some of the rules of the original game:

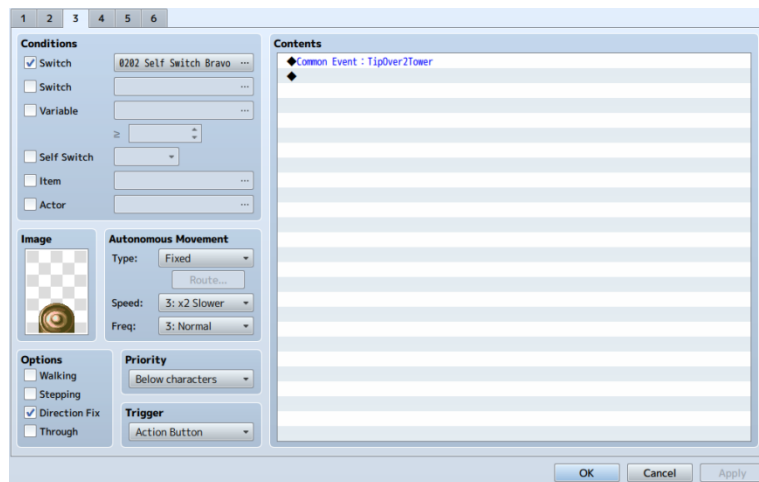
- The player cannot move their piece over gaps between pieces or diagonally, only orthogonally.
- Stacks of pieces may only be tipped over if the player is on the stack, and the player must follow the stack as it falls to its new location.
- Stacks cannot fall onto other stacks, cannot fall onto already fallen stacks or the goal piece, and cannot fall off the board.

The first rule is covered by basic RPG Maker functionality. Outside of plugins and scripted movement, diagonal movement is not available to the player by default. As for ensuring the player doesn't wander around the board or even off the board, we simply place blank Events everywhere the player isn't supposed to go and set their priority to 'Same as Player'.



An Event's priority determines how it 'physically' interacts with the player or other Events on the Maps. Events with different priorities will slip over or under the other, while events with the same priority will be unable to pass through one-another. By having the edge of the game board lined with these blank Events, and having the Events on the game board with this priority when they're not existing in a game piece state ensures that the player can only move from one game piece to another (which themselves are set to a lower priority so that the player walks over them).

For the second and third rule, let's look at the internal logic of what a game piece does when a player decides to tip it over.



Remember how I said Common Events are super useful for avoiding repeating code? Every game piece in each state refers to one of a few Common Events depending on what state it is in, so none of the code is specifically tailored to any specific location on the board; it's all semi-dynamic. Let's take a real look at the internal logic for the code that handles how the two-high yellow stacks tip over onto the board:

```

◆If : Player is facing Right
◆Script : var x = $gameMap.event(this.eventId()).x;
:         var y = $gameMap.event(this.eventId()).y;
:         var z = $gameMap.eventIdXy(x + 1, y);
:         if (z == 0 || z > 36 || this.getSelfSwitchValue(3, z, 201) || this.getSelfSwitchValue(3, z, 202) || this.getSelfSwitchValue(3, z, 203) || this.getSelfSwitchValue(3, z, 204) || this.getSelfSwitchValue(3, z, 205)) {
:             $gameSwitches.setValue(149, true);
:         }
:         var z = $gameMap.eventIdXy(x + 2, y);
:         if (z == 0 || z > 36 || this.getSelfSwitchValue(3, z, 201) || this.getSelfSwitchValue(3, z, 202) || this.getSelfSwitchValue(3, z, 203) || this.getSelfSwitchValue(3, z, 204) || this.getSelfSwitchValue(3, z, 205)) {
:             $gameSwitches.setValue(149, true);
:         }
◆If : TipOver1 is ON
◆Comment : Nothing happens as the tipping is invalid.
◆Control Switches : #0149 TipOver1 = OFF
◆
: Else
◆Comment : Tower tips, switches on adjacent tiles to passable, moves player.
◆Script : var x = $gameMap.event(this.eventId()).x;
:         var y = $gameMap.event(this.eventId()).y;
:         var z = $gameMap.eventIdXy(x + 1, y);
:         this.setSelfSwitchValue(3, z, 201, true);
:         var z = $gameMap.eventIdXy(x + 2, y);
:         this.setSelfSwitchValue(3, z, 201, true);
◆Control Switches : #0202 Self Switch Bravo = OFF
◆Set Movement Route : Player (Wait)
:         : ◀Jump : +1, +0
◆
: End
◆
: End

```

*It took four screenshots to capture this segment, and this is only a fourth of the code for this particular Common Event.*

Let's go from top to bottom. The code is a mix of drag-and-drop and JavaScript, but hopefully it's legible.

```

◆If : Player is facing Right
◆Script : var x = $gameMap.event(this.eventId()).x;
:         var y = $gameMap.event(this.eventId()).y;
:         var z = $gameMap.eventIdXy(x + 1, y);
:         if (z == 0 || z > 36 || this.getSelfSwitchValue(3, z, 201) || this.getSelfSwitchValue(3, z, 202) || this.getSelfSwitchValue(3, z, 203) || this.getSelfSwitchValue(3, z, 204) || this.getSelfSwitchValue(3, z, 205)) {
:             $gameSwitches.setValue(149, true);
:         }
:         var z = $gameMap.eventIdXy(x + 2, y);
:         if (z == 0 || z > 36 || this.getSelfSwitchValue(3, z, 201) || this.getSelfSwitchValue(3, z, 202) || this.getSelfSwitchValue(3, z, 203) || this.getSelfSwitchValue(3, z, 204) || this.getSelfSwitchValue(3, z, 205)) {
:             $gameSwitches.setValue(149, true);
:         }
◆If : TipOver1 is ON
◆Comment : Nothing happens as the tipping is invalid.
◆Control Switches : #0149 TipOver1 = OFF
◆
: Else
◆Comment : Tower tips, switches on adjacent tiles to passable, moves player.
◆Script : var x = $gameMap.event(this.eventId()).x;
:         var y = $gameMap.event(this.eventId()).y;
:         var z = $gameMap.eventIdXy(x + 1, y);
:         this.setSelfSwitchValue(3, z, 201, true);
:         var z = $gameMap.eventIdXy(x + 2, y);
:         this.setSelfSwitchValue(3, z, 201, true);
◆Control Switches : #0202 Self Switch Bravo = OFF

```

This particular segment handles how the tower should tip over to the right, as that's the direction the player is facing. If the player wants to tip the tower over to the left, their player piece should

face left and a different segment of code below this one will handle those calculations (the functions are the same, but the math is subtly different. Same for the up and down directions).

In the first block of JavaScript right under the top If statement, we first note down three variables: X, which stores the horizontal location of where the Event is located on the Map, Y, which stores the vertical location, and lastly Z, which stores the Event Id of the Event to *the immediate right* of this Event. Why do we need the Event Id of the adjacent Event? Because we need to perform a check of what state that Event is in, and the easiest way to do so is if we have that Event's Id. The first If statement written in JavaScript performs that check. It first checks if the Id of the Event is zero, which means that there is no event, and that the tower would be toppling off the edge of the game board, which is an illegal move. It also checks if the Event Id is greater than 36, because the game piece Events Id's are all 1 through 36 (based on the order I placed them on the map), so if the tower is trying to topple on something else that isn't on the board such as one of the barrier Events that keep the player on the board, then that also is an illegal move. The next few conditionals in the If statement check the value of our special self-switches and see if the Event is currently manifesting itself as a game piece. If the If statement returns true, that means that tipping the tower would be an illegal move, and we have a single switch –in this case 0149 'TipOver1' - flip to true (also known as 'ON' in the drag-and-drop code). The second If statement in the first block of JavaScript also performs the same check, but for the spot two spaces to the right of the current Event as the Z value was updated between If statements. If the tower was tipped, it would be occupying two spaces after all, and we need to ensure both of them are free and clear.

Once the JavaScript code is finished running and the value of switch 0149 has either been flipped or left alone, we move onto the next If statement which flows into the first indented segment if 0149 'TipOver1' is true. If that happens, nothing happens as the move is illegal. Switch 0149 is then flipped back off and the Common Event will naturally conclude without anything happening for the player. If however the move the player is attempting to perform is legal, then Switch 0149 will remain off and the Else statement will run. In this case, we enter another block of JavaScript which looks extremely similar to the first block and we grab the same variables as before (technically the x and y variables should still hold their original values as the scope of the JavaScript block doesn't end when the block ends, but when dealing with this hybrid bastardization of code it's better to be safe and less efficient than sorry and crashing to desktop). Using the same values as before, we now flip the empty Events in the spots of consideration to exist as fallen stack pieces, followed by converting the current stack the player is standing on to be an empty void Event. Finally, the code directs the player piece to follow the "falling" stack of pieces and moves them from the now-empty space to the now-not-empty space.

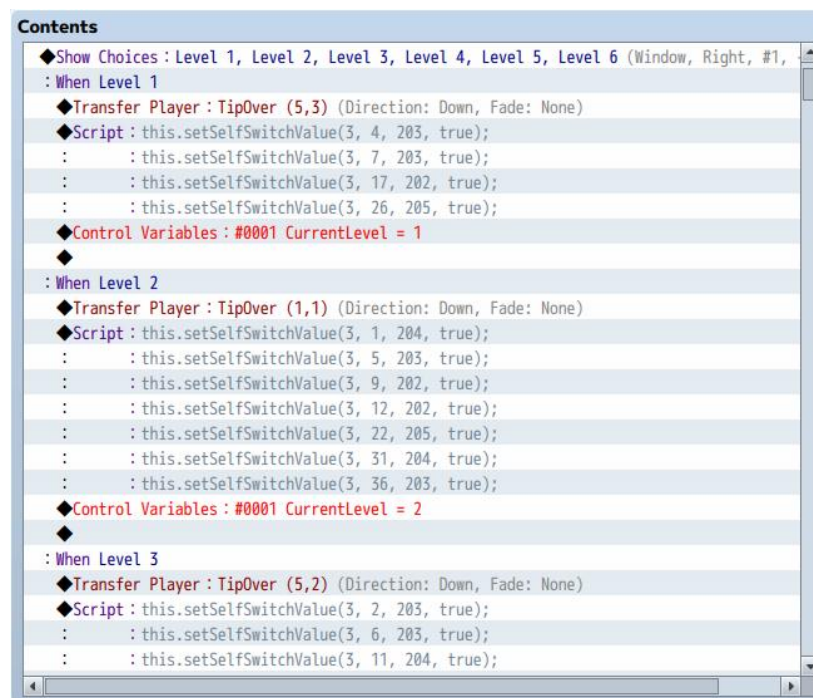
This is the fundamental code for all the game pieces, and it ensures that the second and third simple rules I mentioned earlier are enforced. For topping the tower stacks in different directions, or toppling towers of different heights, the code is mostly the same but with some pluses flipped

to minuses in the math and several more If statements to confirm that all game board spots of consideration are empty and valid for the larger towers to fall over.

## Final Touches:

The previous segment covered the main logic of how the game works while in action. This segment will cover some of the surrounding supporting code.

First, how the game levels are set up. Something has to put those towers on the board in the first place, and it's this script:



Well, script-s. Each level has a unique configuration of pieces, and this is probably the most straight-forward way to set each one up. When the player chooses their level from the menu, the game transfers the player character to the Map where the game board is located and drops them at the starting position. The JavaScript then runs and toggles the state of the appropriate game piece Events by turning on the required special self-switches.

If you've played the game before reading this document and noticed that the pieces don't actually start on the board but rather shuffle onto it like a solitaire deck, that effect was achieved by transporting all of the game piece Events to the corner in a single frame and setting up individual movement routes for each one to jump back to their original spots in sequence.

Before we finish up with this short segment of the document, let's talk about these two buttons at the bottom of the board:





*These two buttons reset the board and returns the player to the level list, respectively.*

Typically there's no custom buttons in RPG Maker MV, this was yet again implemented with a plugin called *YEP Picture Common Events*. It allows you to click a picture and trigger a common event. In this case, the two common events will either reset the game board, or bring the player back to the menu. The latter shares some of its code with the goal piece on the board and it's relatively simple so I won't bother going over it in detail, but the reset button is far more interesting in how it functions.

RPG Maker MV, and probably all of the other versions, allows the player to make saves at will. They open a menu, navigate to the save button, make a save, and continue their game with the assurance that they can reload from their save if they need to. In my main game project that this demonstration program came from, the player can encounter fatal ends to their journey, and very few players like losing hours of their progress because they forgot to make a save. Thus I wrote a script that secretly makes a save for the player right before they cross the point of no return, and whenever the player character perishes I simply have the game reload that save for the player, undoing their fatal mistake and giving them a second chance.

Although this board game has a far less morbid context, the code is able to be reused for the same purpose: rather than writing the logic to reset the board manually, I just used the game's own save system to my advantage to reset the board by going back in time with no interruptions noticeable by the player.

```

◆Script : $gameSystem.onBeforeSave();
:       : if (DataManager.saveGame(56)) {
:       :   StorageManager.cleanBackup(56);
:       : }
◆

◆Script : if (DataManager.loadGame(56)) {
:       :   $gamePlayer.reserveTransfer($gameMap.mapId(), $gamePlayer.x, $gamePlayer.y);
:       :   SceneManager.goto(Scene_Map);
:       :   $gameSystem.onAfterLoad();
:       : }

```

*The Common Event code for saving and loading the game in the background, respectively. While it is more effective than most other solutions to give the player a second chance, using what is essentially time travel to solve the problem feels like cheating.*

# Graphics

If you were reading this document for how the TipOver game itself works, you can stop reading here. Otherwise, I'll be explaining some of the superfluous additions to the project mainly in the way of visuals.

## The Game Master:



*Who is she, and why is her graphic blinking in the game?*

I dabble in 3D animations in my spare time. The graphics for the person above was rendered in a third-party animation program outside of RPG Maker MV, so I won't be going into detail about the steps of that process. I will however explain how I got her graphics working in the game.

If I wanted to just have her static image on the screen, it would have taken less than a minute of work: simply place the picture in the images folder and run the command to show the picture on the screen. Of course, if you're looking at her in the game, you'll notice that she's blinking. So is her graphic an image or a movie? The answer is yes.

By default (again), whenever RPG Maker plays a movie file, everything else stops. The screen darkens and the video plays, ends, and then the game resumes. I call this the "theater" experience. If you want a video to play on the screen in the background without the game stopping

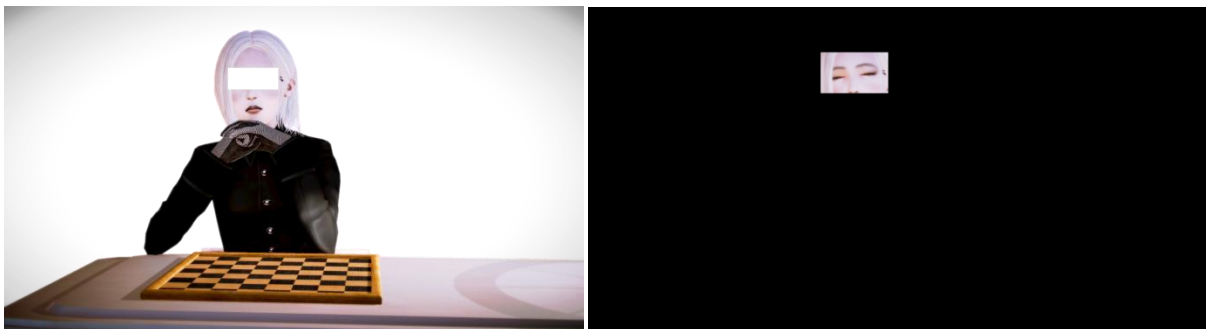
everything else, you (again) have to rely on an external plugin. The one I used in this case is the *YSP Video Player*.

```
Script
ysp.VideoPlayer.loadVideo('WinterBlossomGameBlink.webm')
ysp.VideoPlayer.newVideo('WinterBlossomGameBlink.webm', 4
ysp.VideoPlayer.playVideoById(4)
ysp.VideoPlayer.setLoopById(4)
```

*The script to load, play, and loop the video.*

This would be fine and perfect if I had a full transparent video of the Game Master just sitting and blinking that I could put on loop. To spare you my woes and whining, let's just say that I could produce a single frame with a transparent background and a full video of her blinking, but not a full video of her blinking with a transparent background. Rendering animations properly is hard, okay?

So in the end, I mixed and matched the two. I cropped out the eyes on the transparent picture and cropped out everything *but* the eyes in an external video editor and replaced the cropped out portions in both of them with transparency.



*The white on the left is transparency. The black on the right is transparency.*

As I took both the image and the video from the exact same camera position in the render, both of them will naturally line up on screen almost pixel-perfect as long as an equal aspect ratio is maintained (in this case 16 by 9, or 1366 by 768 pixels to match the screen resolution).

And there we go, we've got a fake blinking person staring at the player. If I had a paid subscription of Adobe Acrobat DC, I probably could even add a clip of it in action here in the document. Unfortunately I don't, so you'll have to go into the game to see it for yourself.

## The Game Overview:



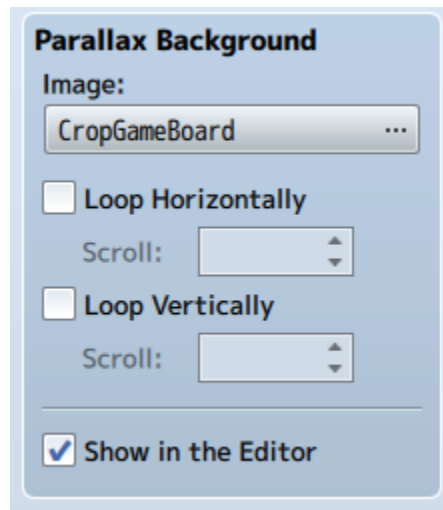
*The shadows took me longer than you would probably be willing to hear me rant on about.*

Same as before, this image was rendered outside of the game. I had to ensure that the game boards from both the render and the game shared the exact same size, position, and dimensions on screen in order for the illusion to work. After the render was completed, I cropped out the checker board and turned it into a parallax background in the game.





In RPG Maker, a parallax background is essentially an image that is rendered underneath everything else where there isn't a Map tile or Event entity image in the way, which makes it good for rendering fake skies or distant cliffs. In fact, the step where I cropped out the 3D game board was probably unnecessary in the end as the image would have rendered underneath the 2D game board anyways, but it was a product of experimentation of different image display options and there's no need to un-crop the image after the fact if the end result is not noticeable to the user.



*If your parallax background has a repeating texture, then the loop and scroll options could be useful.*

## Resources

The following main resources were used in the making of this project.

- RPG Maker MV
  - Plugins:
    - Yanfly's plugin suite: [http://www.yanfly.moe/wiki/Main\\_Page](http://www.yanfly.moe/wiki/Main_Page)
    - Yanfly's Self Switches & Variables  
[http://www.yanfly.moe/wiki/Self\\_Switches\\_%26\\_Variables\\_\(YEP\)](http://www.yanfly.moe/wiki/Self_Switches_%26_Variables_(YEP))
    - Yanfly's Button Common Events  
[http://www.yanfly.moe/wiki/Button\\_Common\\_Events\\_\(YEP\)](http://www.yanfly.moe/wiki/Button_Common_Events_(YEP))
    - YSP Video Player  
[https://forums.rpgmakerweb.com/index.php?threads/ysp\\_videoplayer.83758/](https://forums.rpgmakerweb.com/index.php?threads/ysp_videoplayer.83758/)
    - Terrax Lighting <https://forums.rpgmakerweb.com/index.php?threads/terrax-lighting-system.49339/>
    - Yanfly's Picture Common Events  
[http://www.yanfly.moe/wiki/Picture\\_Common\\_Events\\_\(YEP\)](http://www.yanfly.moe/wiki/Picture_Common_Events_(YEP))

- HimeWorks Large Choices <https://himeworks.com/2015/10/large-choices-mv/>
  - Yami Skip Title <https://forums.rpgmakerweb.com/index.php?threads/missing-plugins-download.46479/>
- Graphics
  - Avery's Chess and Checkers set <https://forums.rpgmakerweb.com/index.php?threads/avys-mv-stuff.53317/page-76>
  - 2D character generator parts: <https://www.hiddenone-sprites.com/mv-generator-parts.html>
- Music
  - <https://rpgmaker.net/musicpack/>
- Shotcut video editor
- GNU Image Manipulation Program (GIMP)