The RPG Series
RPGs are one of gaming's most popular and longest lived genres, and some of the most popular series in the world have had entries (or even got their start) on the GameBoy. And now, thanks to GB Studio, it's your turn to create an RPG of your own!

In this series, we will cover how to use GB Studio's built-in events to create a basic RPG system that you can expand on for your own game. All project files will be available to download at the end of the article, and inside you'll have all the examples and assets we use in each article (along with some additional comments). You can use these as a reference or even expand on them to create your own game.

Quest 1: Battle System Sequencing & Basic Actions
While it may not seem obvious at first, GB Studio has everything you need to build an RPG Battle System - you just need to do the heavy lifting. In our first quest, we'll build out the core structure of our battle: a turn sequence for a single player character and single enemy, and allow the player to make a choice between two actions.

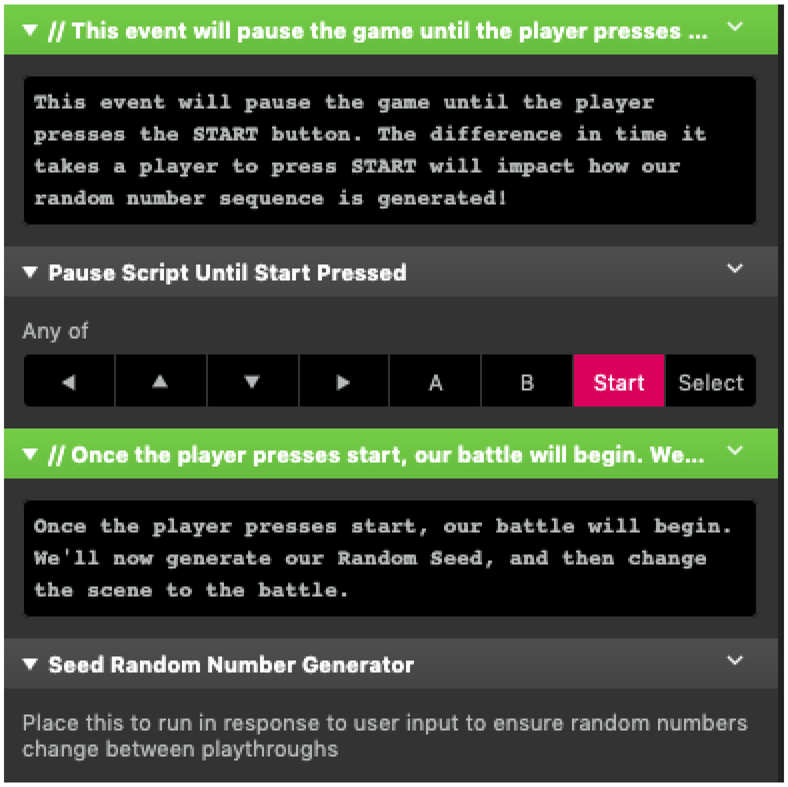
For today's quest, we're going to work on building these with two scenes: a **Boot Scene** and our **Battle Scene**. Let's get started!

The Boot Scene
In a normal game, your player would enter a battle from an overworld or other scene, and we would use those to control the events we would need to run before a battle. Since we don't have that in our tutorial yet, the Boot Scene will be where we run the events that will be handled elsewhere in a full game. Don't worry - as the tutorial goes on, we'll introduce a basic overworld where we can move some of these scenes, but for now we'll be using this placeholder.

The most important event for our game that is necessary to run in our Boot Scene is the **[Seed Random Number Generator]** event. Since our battle system will be using random numbers to determine if our various attacks, spells, and other actions pass or fail, we'll need to make sure our game is set up to use these correctly.

If you're familiar with GB Studio already, or have been keeping up with other articles on our site, simulating randomness can be tricky! In order to prepare our game properly, we need to *seed* the random number generator, and in order to make sure this is different for every time we play our game, the most basic way to do this is to run the event after some kind of player input.

For our game, we'll do this in our **Boot Scene**. In the **On Init** script area, we'll use the **[Pause Script Until Button Pressed]** event to make our game wait until the player presses START, then place our **[Seed Random Number Generator]** event after that. Now, our game will generate a different sequence of numbers based on the time the player started our game. Simple, but effective!


▼ // This event will pause the game until the player presses ...

```
This event will pause the game until the player
presses the START button. The difference in time it
takes a player to press START will impact how our
random number sequence is generated!
```

▼ Pause Script Until Start Pressed

Any of

◀ ▲ ▼ ▶ A B [✓ Start] Select

▼ // Once the player presses start, our battle will begin. We...

```
Once the player presses start, our battle will begin.
We'll now generate our Random Seed, and then change
the scene to the battle.
```

▼ Seed Random Number Generator

Place this to run in response to user input to ensure random numbers change between playthroughs

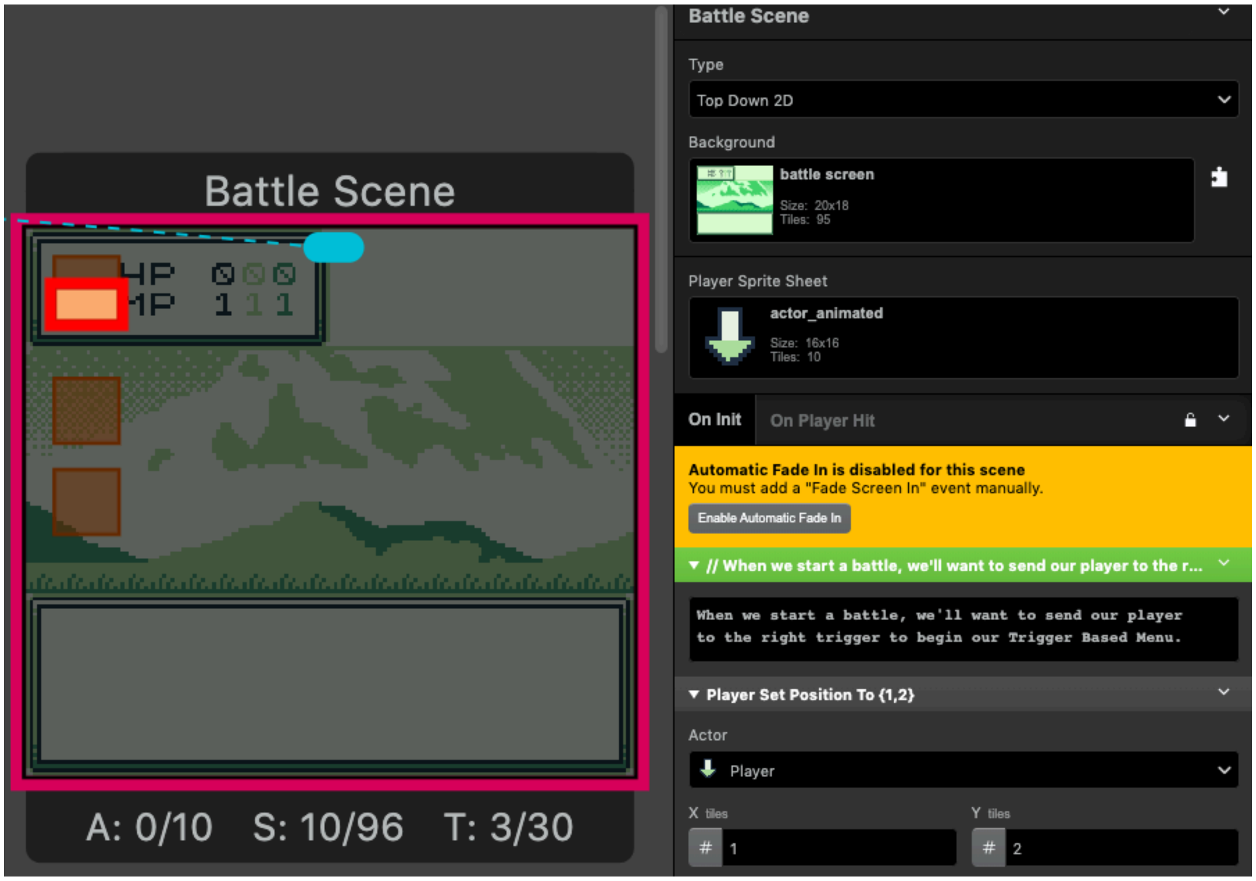
As we build out our system, there will be more we'll want to run before a battle starts, but we'll move on for now.

The Battle Sequence

Not every RPG is built the same, and even the earliest RPGs on the GameBoy all had major differences in their battle systems; classics like Dragon Quest, Final Fantasy, and Pokémon all have very iconic battle systems, and each system has their fans. At their core though, these turn-based battle systems all have similar DNA. While our tutorials are going to provide a structure that resembles the Dragon Quest (with a "forward facing" battle screen and menus), many of the concepts presented here can be extrapolated into other styles.

At their core, an RPG battle is just a menu. Players are provided choices, and based on their selections, various events will occur that determine if you win or lose. To build these sequences and provide our player with the menus needed to make their choices, our battle system will be built using something we've covered in an earlier article: trigger based menus!

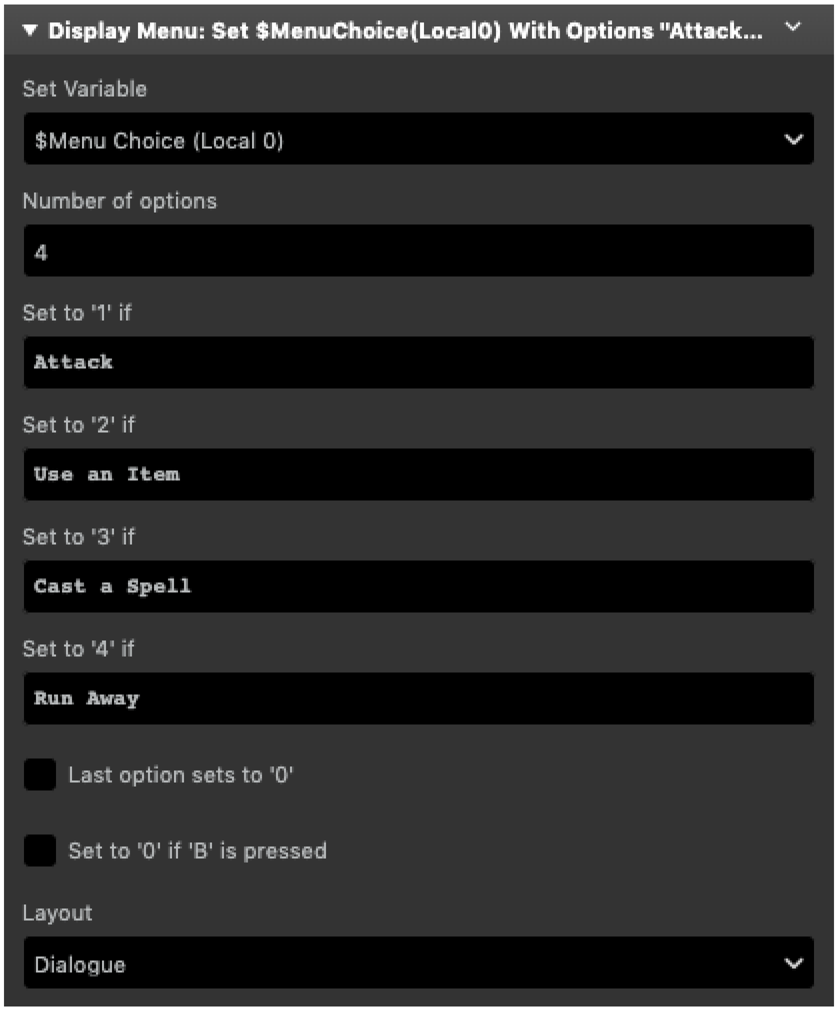
To start us off, in the **On Init** area of our Battle Scene, we'll want to use the **[Set Actor Position]** event for our player actor and send them to the coordinates our first menu trigger. This trigger will be our "Player Turn" trigger, and handle our first menus.



```
When we start a battle, we'll want to send our player
to the right trigger to begin our Trigger Based Menu.
```

(Note: In later articles, we'll create a system where the enemy can take a turn before the player, but for now the player will always go first.)

In our first trigger (renamed "Player Turn" in our project), we'll add a [Display Menu] event. This will let us provide the player with up to 8 choices, but for our game we'll only be using 4. Increase the number of options and give them a name!



Display Menu: Set $MenuChoice(Local0) With Options "Attack...

Set Variable: $Menu Choice (Local 0)

Number of options: 4

Set to '1' if: Attack

Set to '2' if: Use an Item

Set to '3' if: Cast a Spell

Set to '4' if: Run Away

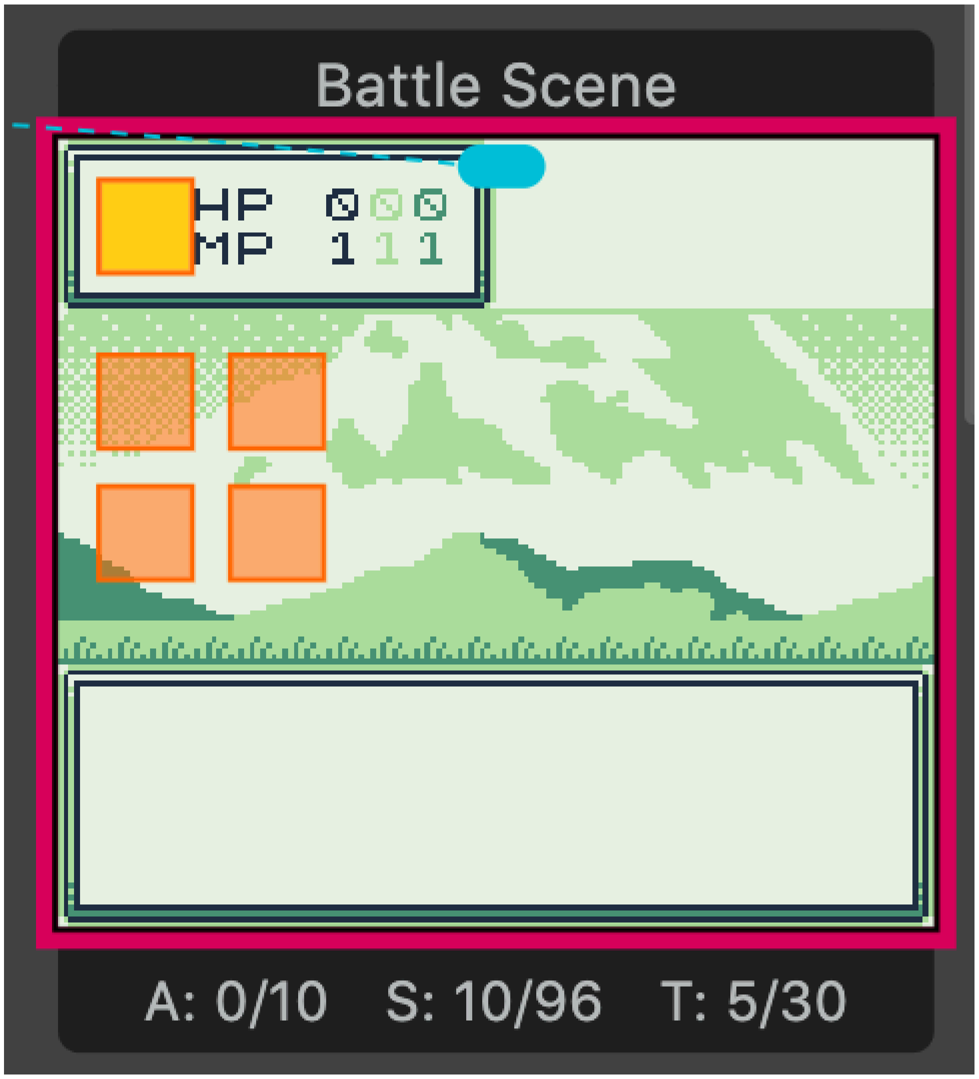
☐ Last option sets to '0'

☐ Set to '0' if 'B' is pressed

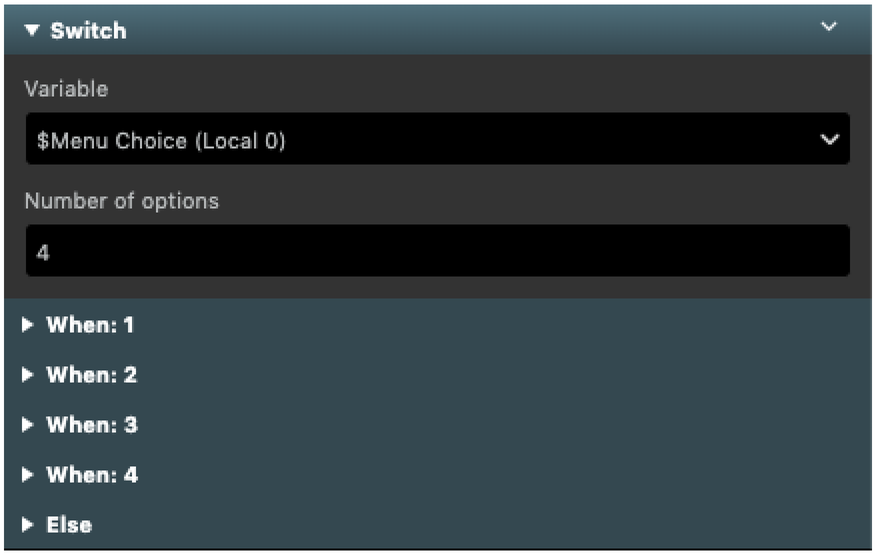
Layout: Dialogue

Deselect the “Set to 0 if B Pressed” option as well; this will prevent the player from canceling their input. A menu also needs to save its choice to a variable; in our case, we’ll use Local 0.

Now that the player has their options, we need to create a way to handle them. Create four more triggers in your Battle Scene, and name them based on the four possible choices.



Then, back in our Player Turn trigger, add a new **[Switch]** event after our menu event. A Switch is a convenient way to handle multiple possibilities from a single event. We'll set our switch to be based on Local 0, and give it four possible values: 1 - 4, just like our menu.


Switch

Variable
$Menu Choice (Local 0)

Number of options
4

When: 1

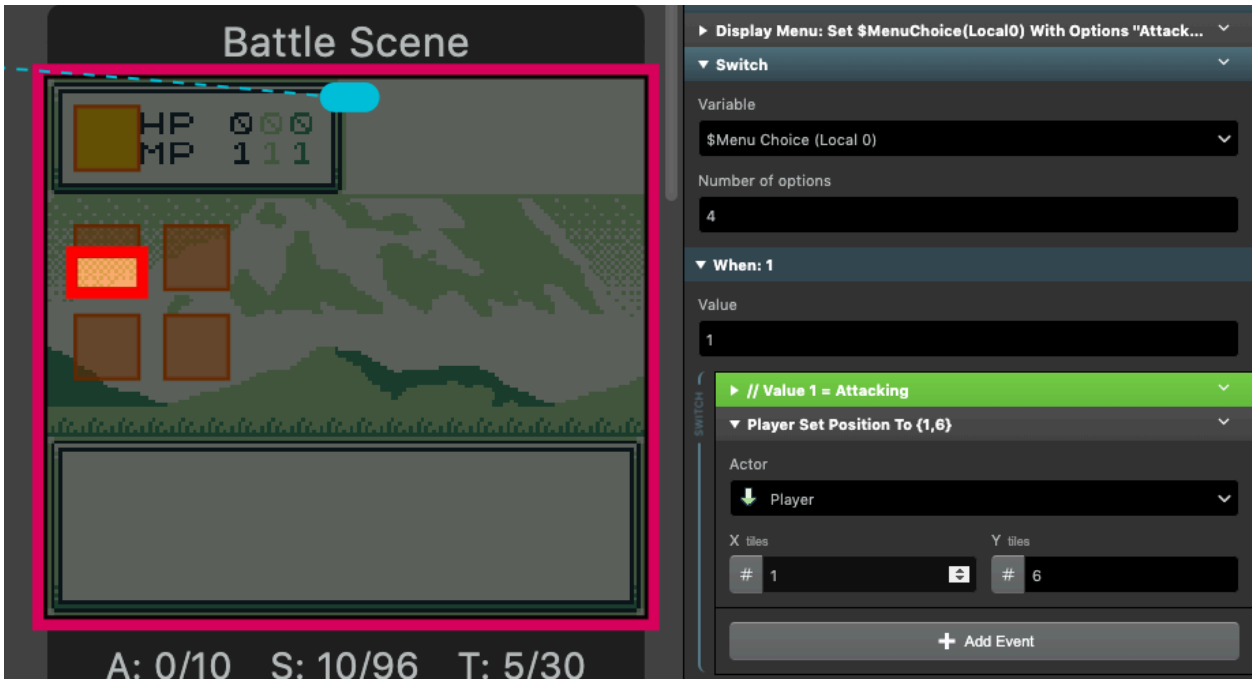
When: 2

When: 3

When: 4

Else

For each value, we'll want to add another [Set Actor Position] event that moves the player to the trigger that handles that sequence. For example, if we selected "Attack" (a value of 1), the "1" section of our switch will send the player to our "Attack" trigger.



Display Menu: Set $MenuChoice(Local0) With Options "Attack...

Switch

Variable
$Menu Choice (Local 0)

Number of options
4

When: 1

Value
1

// Value 1 = Attacking

Player Set Position To (1,6)

Actor
Player

X tiles
1

Y tiles
6

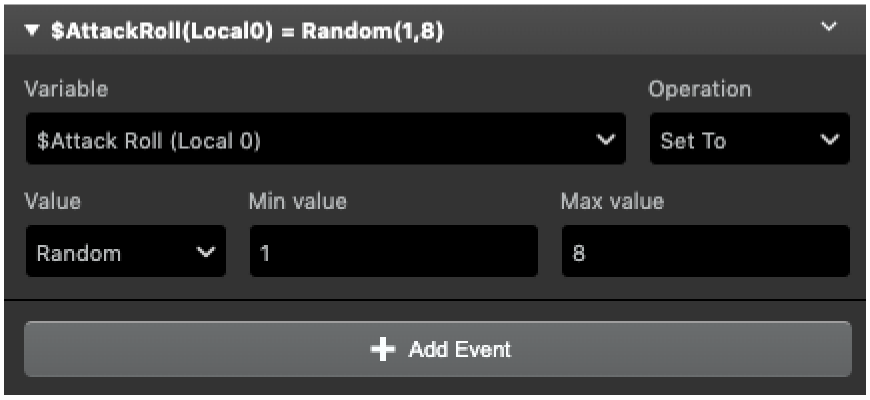
Add Event

Once we've done this for all four, we'll set up a sequence for each option.

Fight Trigger
We'll need to attack a monster in order to defeat it, but we don't want to make it too easy for ourselves. Since we went through all the trouble of seeding our random number generator, let's make a simple dice roll to determine if we hit or missed the monster. In order to do this, we'll

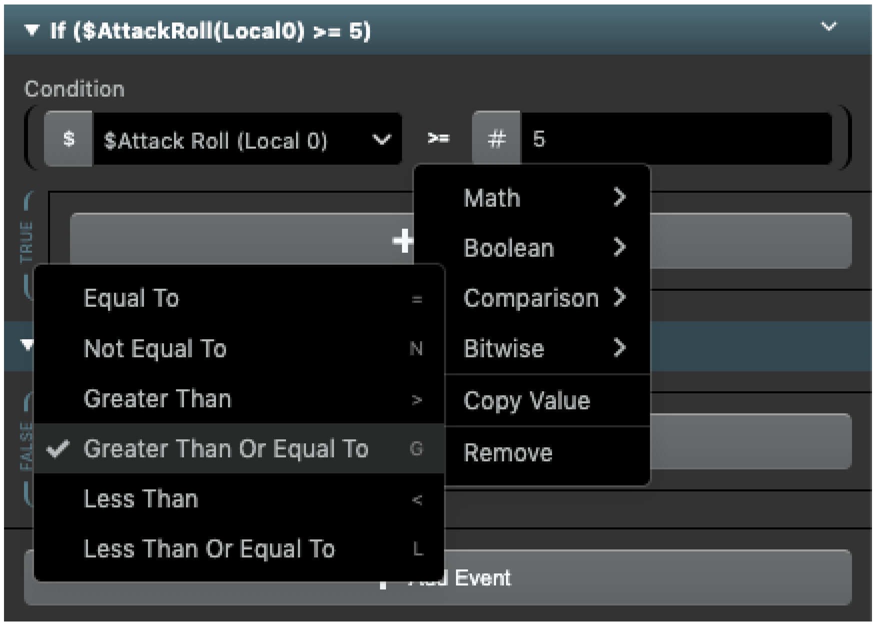
need to use the power of math!

In our Fight trigger, add a **[Math Functions]** event, setting the Local 0 variable to a random number between 1 and 8, like so:


$AttackRoll(Local0) = Random(1,8)

Variable: $Attack Roll (Local 0)
Operation: Set To
Value: Random
Min value: 1
Max value: 8

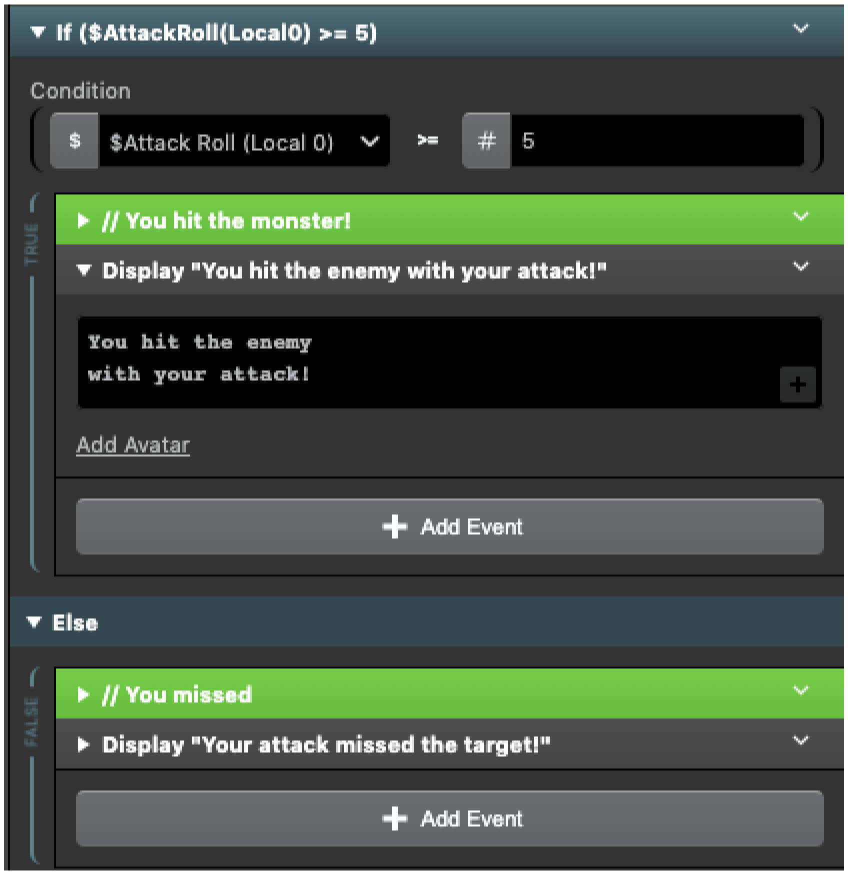
Then, we'll want to use a **[If Compare Variable to Value]** event to check if we passed or failed. Let's make it a 50/50 shot to hit the enemy. If our Variable is greater than or equal to 5, we'll pass. Otherwise, we'll fail.


If ($AttackRoll(Local0) >= 5)

Condition: $ $Attack Roll (Local 0) >= # 5

Note: If you're returning to GBStudio, the compare event interface is a little different in 4.0! To set this up, you'll want to click the == sign between the two condition boxes, then select "Comparison" to find our options.

As we flesh out our game, we'll add ways to actually deal damage. For now, let's continue building out our sequence flow. You can add some dialogue to show the player if your attack hit or missed using **[Display Dialogue]**.



If ($AttackRoll(Local0) >= 5)

Condition

$Attack Roll (Local 0) >= 5

// You hit the monster!

Display "You hit the enemy with your attack!"

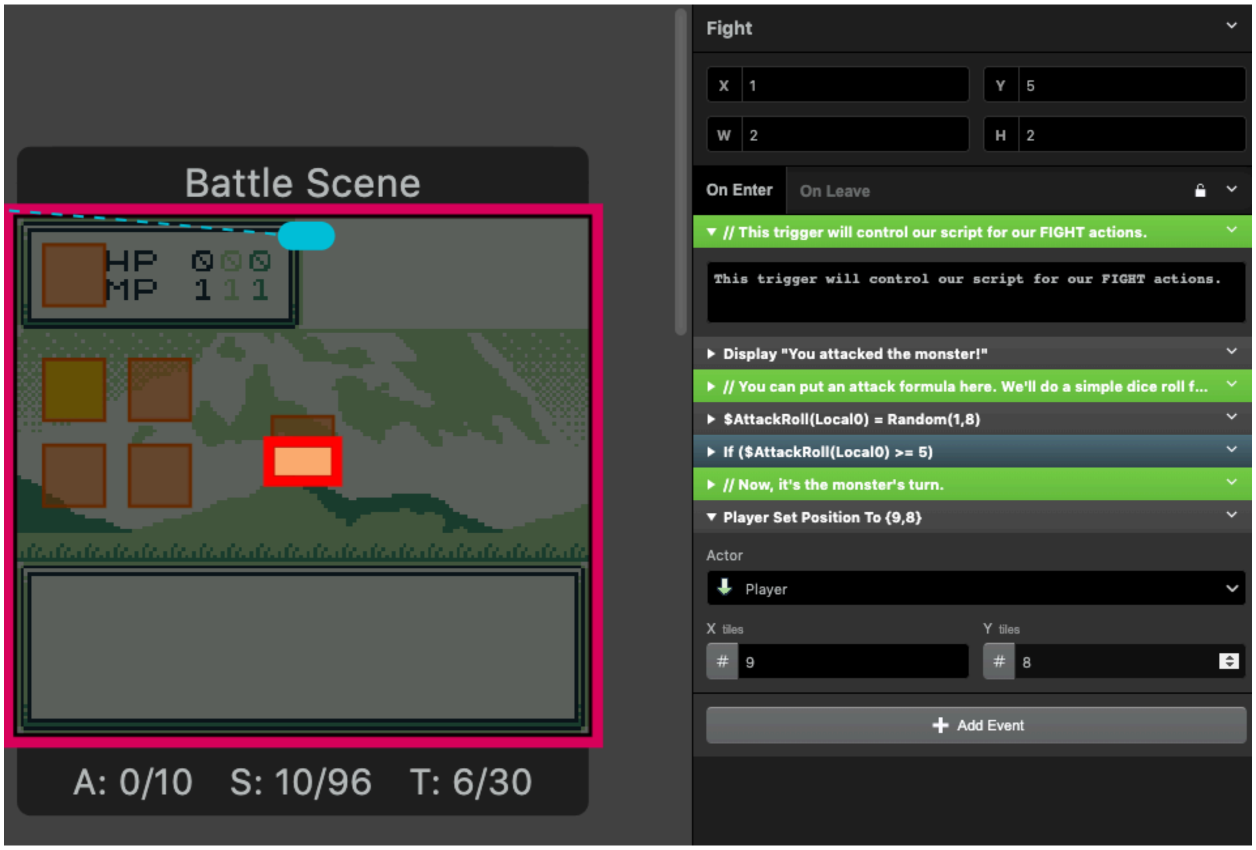
```
You hit the enemy
with your attack!
```

Else

// You missed

Display "Your attack missed the target!"

After that, it's going to be the enemy's turn. Create a new trigger, and then use the **[Set Actor Position]** to move the player to its coordinates.



Fight

X 1 Y 5
W 2 H 2

On Enter | On Leave

// This trigger will control our script for our FIGHT actions.

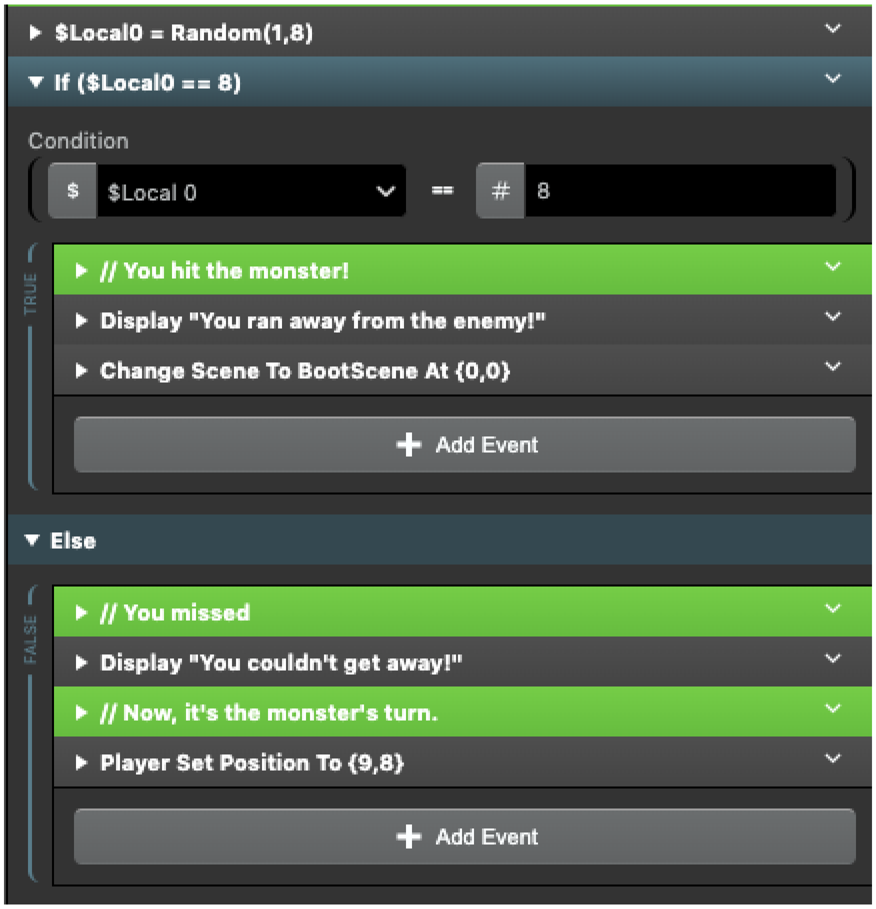
```
This trigger will control our script for our FIGHT actions.
```

Display "You attacked the monster!"
// You can put an attack formula here. We'll do a simple dice roll f...
$AttackRoll(Local0) = Random(1,8)
If ($AttackRoll(Local0) >= 5)
// Now, it's the monster's turn.
Player Set Position To (9,8)

Actor: Player
X tiles: 9
Y tiles: 8

Flee Trigger

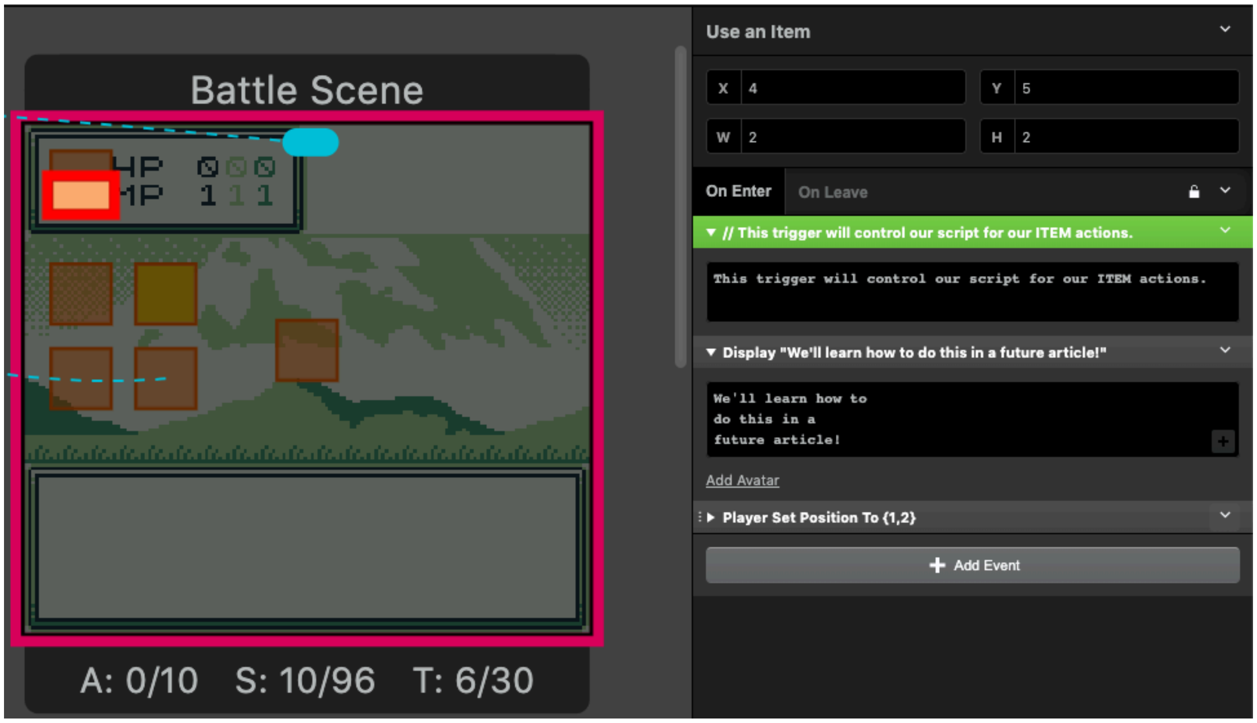
In our Flee event, let's do something similar. Create another [Math Functions] event, and then add a [Compare Variable to Value] event - but this time, we'll make it harder to run away than attack the monster - let's make it a 1 in 8 chance to get away!



```
▶ $Local0 = Random(1,8)
▼ If ($Local0 == 8)
  Condition
  ( $ $Local 0  ==  # 8 )
  TRUE
    ▶ // You hit the monster!
    ▶ Display "You ran away from the enemy!"
    ▶ Change Scene To BootScene At (0,0)
    + Add Event
▼ Else
  FALSE
    ▶ // You missed
    ▶ Display "You couldn't get away!"
    ▶ // Now, it's the monster's turn.
    ▶ Player Set Position To (9,8)
    + Add Event
```

Normally, if we passed, we'd return to our overworld map. For now, we'll just send the player back to the Boot Scene. But if we fail, it's the monster's turn, and we'll send them to our monster trigger!

Using Items and Casting Spells Triggers
We'll cover our more advanced actions in a future article. For now, if a player selects either of these, we'll have their trigger's **[Set Actor Position]** back to the Player Turn trigger, starting our menu sequence over.



▼ // This trigger will control our script for our ITEM actions.

```
This trigger will control our script for our ITEM actions.
```

▼ Display "We'll learn how to do this in a future article!"

```
We'll learn how to
do this in a
future article!
```

▶ Player Set Position To {1,2}

You can do a similar placeholder event for the Casting Spells trigger for now, too.

The Monster's Turn
On the Monster's turn, they'll get to take their own actions…but we'll find out what they do in our next article!

Set an event to send the player back to the Player Turn trigger, and we'll have successfully created a looping sequence of menu choices. We now have the bones of our very own, albeit very basic, turn-based RPG system!

https://youtu.be/GxYdcUqkj5E

Up Next:
In our next part, we'll create some basic behaviors for our monster, just like our player, and start tracking our first stats: HP and damage values!

Project Files:
You can download the project files used in this article here.