

# Introduction to GameMaker: Workshop 2

## ITCS 4230/5230

### Learning Outcomes

By the end of the workshop the student will be able to:

1. Implement the game objects and mechanics necessary to change rooms
  2. Write Cheat codes
  3. Implement functionality to enable player characters to move objects
  4. Use gravity
  5. Use tiles to create background effects
  6. Implement collisions that take into account continuous movement
  7. Apply basic AI concepts such as
    - a. Enemies that have a patrol path
    - b. Enemies that follow the player
  8. Modify an object's appearance to reflect what it is doing
- 

### Setup

1. Navigate to the **GameMaker: Platform Games** Module in *Canvas*
  2. Download the activity file named **Simple\_Platformer.zip** from the assignment page. This file contains a partially completed game that you will use to learn about platformer games and use as the basis to implement a simple platformer that you will submit.
  3. Unzip the file to an appropriate location on your computer.
  4. Start GameMaker and open the project (Simple\_Platformer).
  5. Make sure the game builds and runs.
-

# Introduction

In this workshop, you will be using a partial game. This partial game is the basis to implement a simple platformer. The game you will submit is similar to games like *Super Mario Brothers* or *Metroid*. However, what you will produce is much simpler.

As you look at the project code, keep in mind that GameMaker Studio uses the Cartesian plane for coordinates, as well as degrees (0-360) for most anything labeled “direction.” a quick reference for these is shown below.

**Direction:**

0 to the right  
90 straight upwards  
180 to the left  
270 straight down

**Coordinates:**

x+ moves to the right  
x- moves to the left  
y+ moves down  
y- moves up

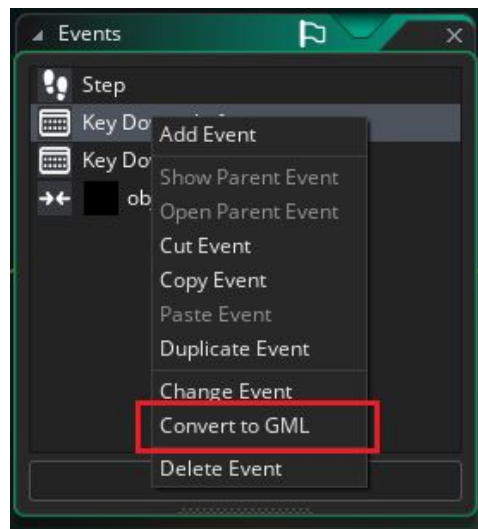
(Note that since -y is the upward direction, the traversal from 0° to 360° is counter-clockwise.)

---

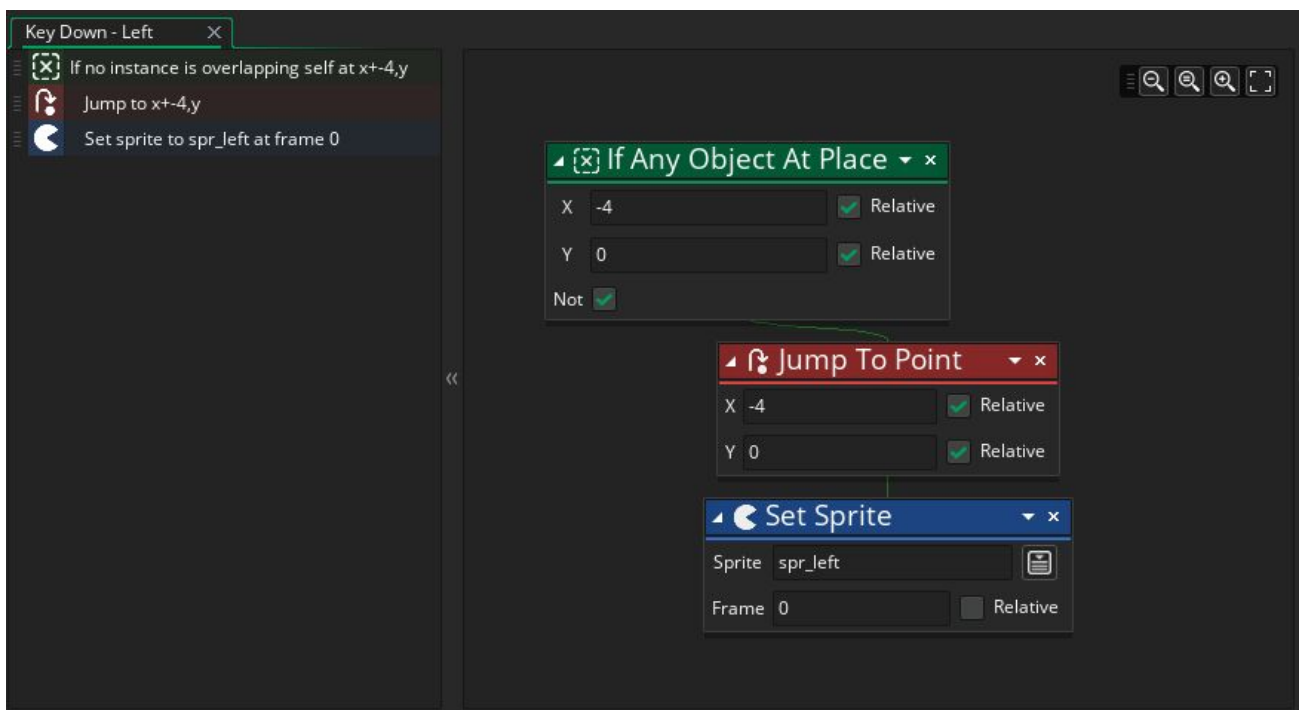
## Part 0 (Optional): GameMaker Language

Even though GameMaker's drag and drop (DnD) functionality may be sufficient for many games, you may want to use programming for your group projects. Hence, it is greatly recommended that you familiarize yourself with GameMaker Language (GML).

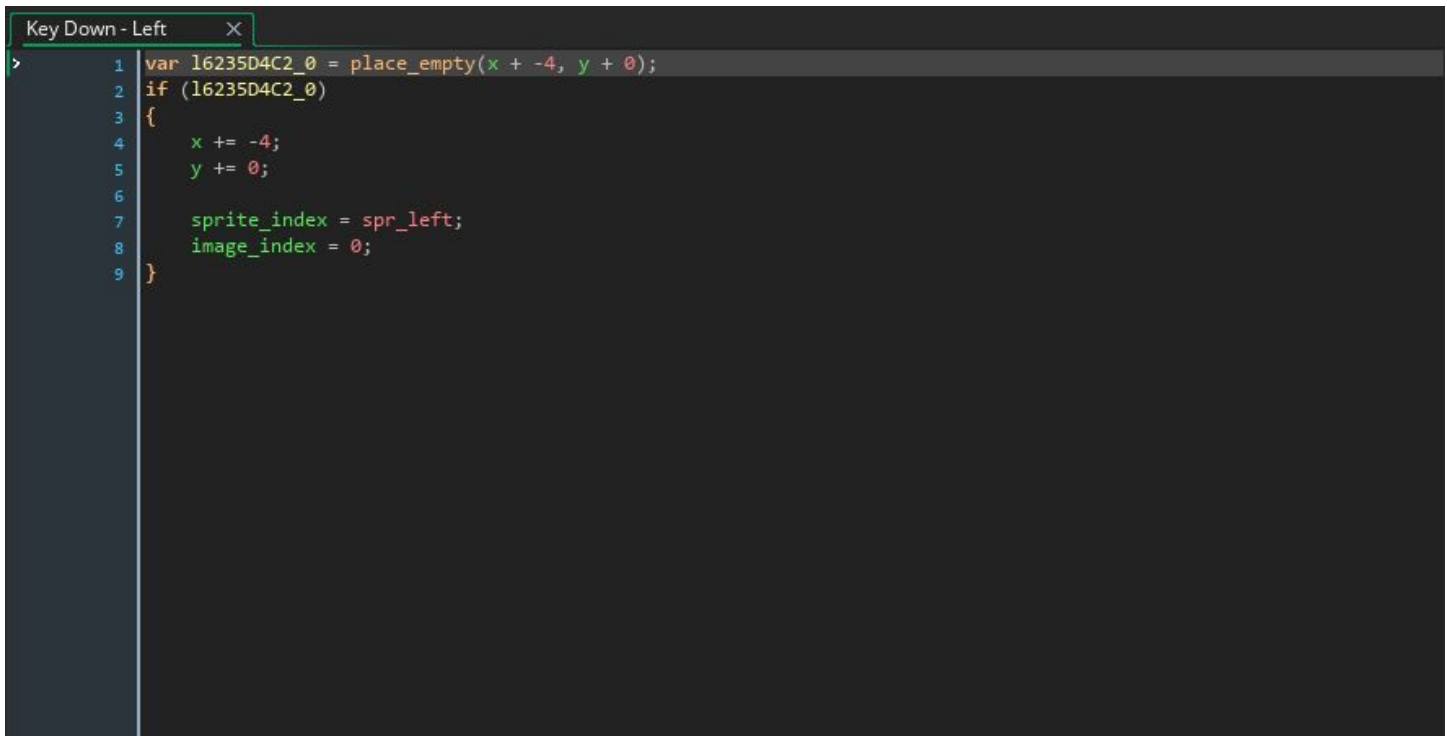
A good way to get started is to right click an event from any object and select the option, **Convert to GML**. This will take the Drag & Drop blocks in the event and convert them into code, which gives you a quick & easy look at GML syntax, as well as what functions GameMaker has to implement the functionality you need.



For instance, the Step Event in *obj\_character* from *Simple\_Platformer* converts from the following Drag and Drop actions:

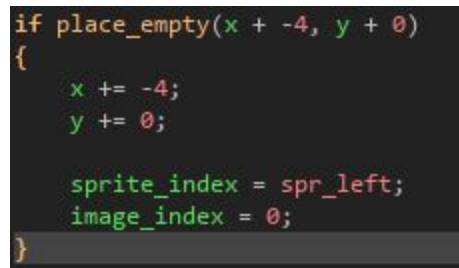


To the following GML code:



```
1 var 16235D4C2_0 = place_empty(x + -4, y + 0);
2 if (16235D4C2_0)
3 {
4     x += -4;
5     y += 0;
6
7     sprite_index = spr_left;
8     image_index = 0;
9 }
```

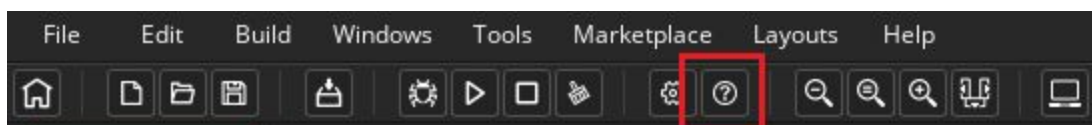
Note that the variable name produced at line 1 is merely a consequence of how GameMaker converts from Drag & Drop to GML. The code can be modified to improve clarity and maintainability, as shown below:



```
if place_empty(x + -4, y + 0)
{
    x += -4;
    y += 0;

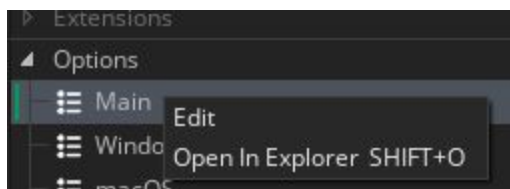
    sprite_index = spr_left;
    image_index = 0;
}
```

For these GameMaker Workshops, your only requirement is that your submitted game has all of the functionality that has been described. How you create that functionality is entirely up to you (though, following the instructions certainly helps). You can use Drag & Drop, GML or even a combination of the two. If you do choose to work with GML, it's recommended that you use this "Convert to GML" technique to get started and refer to GameMaker's built-in manual. The manual can be accessed via **Help -> Open Manual** or by clicking the Question Mark icon near the top of the program.

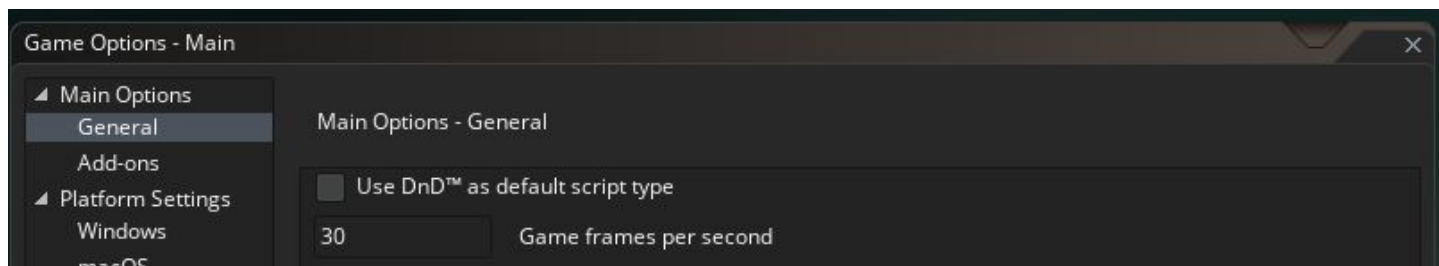


If GameMaker runs slow on your computer, you may have better luck using the online manual, found at <https://docs2.yoyogames.com/>

You may decide that you are more comfortable with GML overall than you are with Drag & Drop. If you are confident in using GML, another way you can convert to GML is to go change **default script type** in the **Options** dialog. In the **Resources** menu, expand **Options**, right click on **Main** and select **Edit**.



From here, **uncheck** *Use DnD as the default script type*, then select apply. This means that when you add a new event, Game Maker will automatically open the event in GML. Note: This will not automatically convert old Drag & Drop boxes to GML, it will only convert new events.

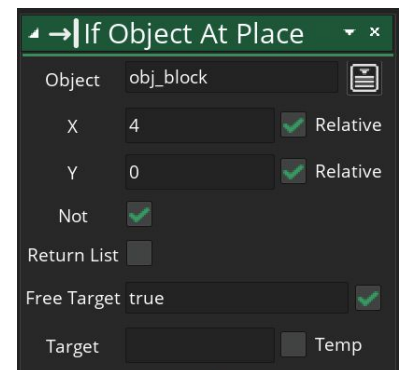


# Part 1: Movement in a Platform Game

In this part of the assignment, you will practice how to implement the basic mechanics of jumping and moving on platforms. This game also provides practice on manipulating sprites to match what the player is doing. **Note that these instructions assume that you have completed the GameMaker tutorials that were assigned as prep work, as well as Workshop 1.** It is also recommended that you refer to the GameMaker Studio Online Help as needed (see *Help* menu → *Open Manual*).

## Objectives:

- Walking left & right
    - Updating character's sprite when walking
  - Jumping
  - Falling down when in midair
    - Limiting maximum vertical speed
  - Avoiding intersecting with walls
- 
- Task 1 - Walking left & right
    - **All the code will be written in `obj_character`.**
    - All of the sprites needed have been provided.
    - Motion in the horizontal direction has already been implemented. Note that the character's sprite always faces right, regardless of the direction of motion. You need to implement the code to correct this, i.e., **the sprite should change to match the direction in which the character is moving.**
      - Change the *Key Down - Left* and *Key Down Right* events to include a **Set Sprite** action that updates the player's sprite to either `spr_left` or `spr_right` depending on the direction of motion.
    - Additionally, we want to make sure that the character can only move if there are no obstacles, use the If Object At Place action to check before updating the value of the character's `x` coordinate. In other words, always check to see if the position is empty before jumping to it!.
  - Task 2 - Jumping
    - Jumping applies vertical momentum - you'll need to set `vspeed` instead of using jump-to-position
      - Remember, upward is negative in the y-direction
    - Run the game and try to jump while still in the air.
    - The character should only be able to jump while they are standing on the ground or on a platform. To address this, add code to check if there is ground *directly below* the player (`x+0, y+1, relative`). This needs to happen before setting the vertical speed.
      - Useful DnD actions: **Any Object at Place** or **Object at Place**
      - Useful GML functions: `place_meeting()` or `place_empty()`
    - You may notice that once `obj_character` jumps it never comes back down...



- Task 3 - Gravity
  - A Crash-Course on Gravity:
    - **gravity** is a built-in variable present in all *GameMaker* objects. It has an accompanying variable, **gravity\_direction**
    - Every game tick, **gravity** adds speed in a specific direction, specified by **gravity\_direction**
    - The amount of change enacted by gravity is dependent on **gravity**'s value - if an object's **gravity** = 0, nothing will happen.
    - **To create a force that pushes obj\_character downwards, set gravity\_direction = 270.**
  - **gravity\_direction** only needs to be set once - that can be done in *obj\_character*'s create event
  - Gravity is usually handled in the **Step** event, which runs every frame
  - Similar to jumping, we need to check whether there is solid ground below *obj\_character*. If there is, **gravity** should be 0, otherwise it should be a positive number (0.5 or 1 works pretty well)
  - After implementing gravity, you should see that after jumping *obj\_character* falls back down, *and then proceeds to fall through the floor...*
- Task 4 - Colliding with walls
  - When walking left/right, we avoid colliding with walls entirely using the conditional check; We're working with **vspeed** to facilitate jumping, which makes *avoiding* collisions much harder. Instead, we can use a collision event to react when a collision occurs.
  - Note that there are 3 wall objects: **obj\_block**, **obj\_blockv** and **obj\_blockh**.
    - **obj\_blockh** and **obj\_blockv** are children of **obj\_block**, so if *obj\_character* has a collision event that activates with **obj\_block**, it will activate with either of them as well.
  - What we need to do here is to set **vspeed** to 0 when colliding with a block, and *obj\_character* will stop falling once they hit the ground.
  - **Additional Note:** if you check **obj\_block**, you'll see that it's flagged as *solid*. What this means is that when *obj\_character* collides with a block, it will be pushed out of the block when the collision event occurs (this ONLY happens if the object has a collision event). If you turn off solid on **obj\_block**, you'll see that upon falling back down to the ground, *obj\_character* will often embed itself into the ground, making left/right movement impossible.
- Task 5: Limiting Vertical Speed
  - If *obj\_character* falls for a long time, their **vspeed** can get to be pretty fast. This can create problems in your game; at best, it might prove detrimental to how your character controls midair (a requirement for platformers), while at worst your character might fall so fast they bypass the floor's hitbox entirely! The solution is an upper limit to **vspeed**.
  - To address this, add a conditional statement in the **Step** event, which checks if **vspeed** is greater than a certain number (12 should work, but we recommend testing).
    - If **vspeed** is greater than 12, set it to 12.

- If you are using GML, GameMaker provides a few handy math functions such as **min()**, which returns the smallest value out of the ones you provide. It might be easy to see how that could be applied here...

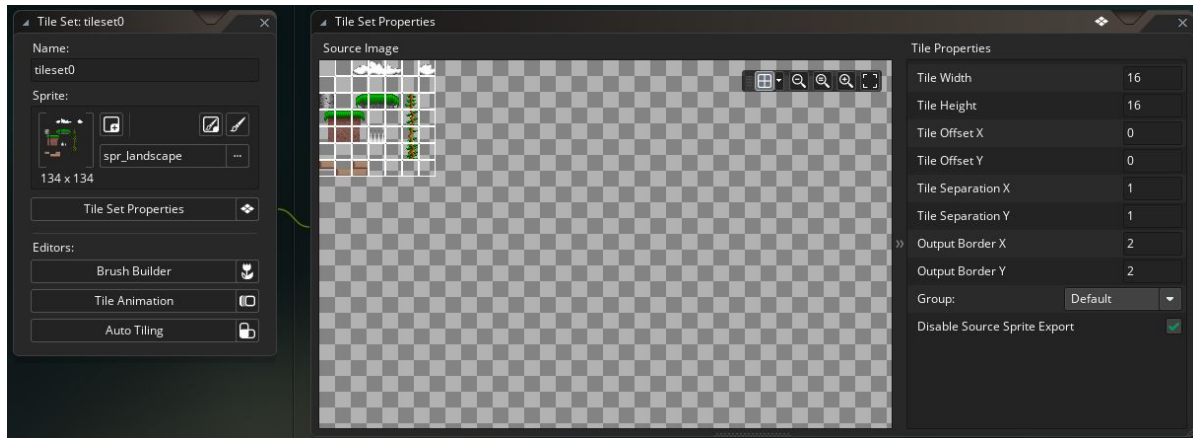
## Task 6: Level Construction

- o The default size of **obj\_block** is determined by its 32x32 sprite. However, GameMaker allows you to resize instances once they're placed in the room editor. This allows you to cover large stretches of level (room) using just one wall object.
  - Scaling an instance in the room editor will alter its builtin variables **image\_xscale** and **image\_yscale**. Since collisions take these variables into account, colliding with a scaled object functions as usual.
  - These changes to **image\_xscale** and **image\_yscale** are made before the object's create event is called. You may be able to find some use for this...





- o You may have noticed that while **obj\_block** and its children have been used as colliders for our level, they are not visible when the game is running. Their **Visible** property is set to *false*, as we have a *tileset layer* providing the visual aspect of our level.
- o *Tilesets in GameMaker* are sprites broken up into equal-size sections, which can be placed on a tileset layer. By default they are purely visual, though there are some advanced functions that can make clever use of them.



- o Open the resource for **tileset0** and examine its properties. The sprite it uses is `spr_landscape`, and it splits the sprite up into 16x16 tiles. *It is recommended you read about tilesets' other properties in the [GameMaker manual](#).*
- o **Note: The top-left tile in a tileset will always be the “empty” tile. If you make a sprite for the purposes of a tileset, know that any image data in the top-left tile space will not be used.**
- o Note that some instances of **obj\_block** are not expanded to the correct size. Use **image\_xscale** and **image\_yscale** to match these instances with the corresponding tileset.

## Part 2: Enemies

This part of the project focuses on adding the enemy characters **obj\_monster** & **obj\_flyer**. Initially, we will be taking a look at some rudimentary enemy behaviors, as well as some potential roadblocks when using inheritance. After that we will sprinkle in some additional trinkets and hazards that may appear in a platformer.

### Objectives for Part 2:

- Defining boundaries for **obj\_monster**
  - Inheriting **obj\_monster**'s behavior through **obj\_flyer**
  - Squashing enemies
  - Death spikes
  - Bonus items
- 
- Task 1 - Giving **obj\_monster** Boundaries
    - Take a look at **obj\_monster** and you should notice that it already has some basic functionality. In its **Create** event, its **hspeed** is set to 2, and upon collision with any of our wall objects, the monster turns around.
      - The effect: our monster walks to the right, and upon hitting a wall it walks to the left. Repeat.

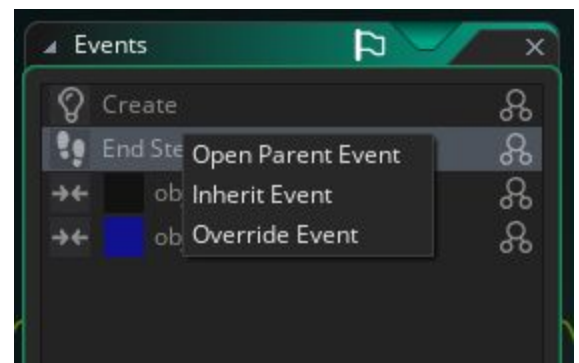
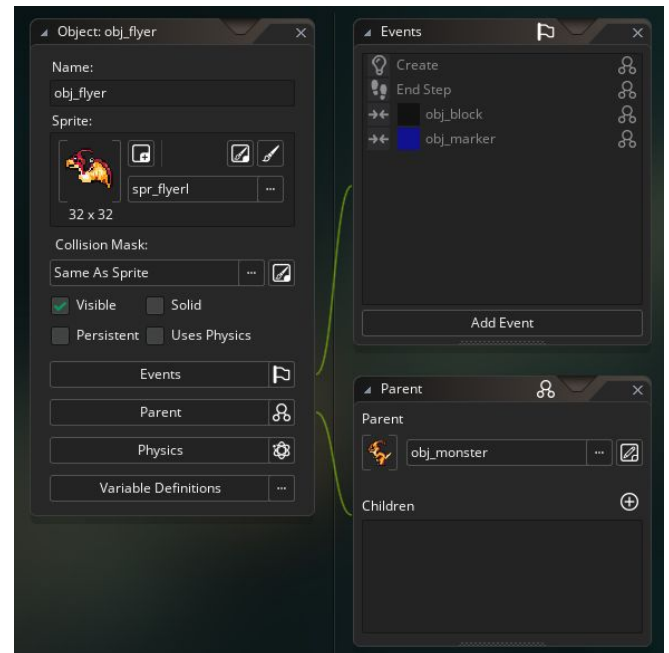


- You'll see, however, that nothing's stopping this monster from walking right off of platforms (and nothing's making them fall down). For our purposes, rather than dealing with gravity we will just prevent **obj\_monster** from walking off the platforms.
- A simple fix for this comes in the aptly-named **obj\_marker** that already exists in the project. This marker is already set to be invisible, so it is perfect to set boundaries.
  - **Duplicate the event that makes our monster turn around on walls, and have it do the same upon collision with **obj\_marker**.** Right-click on the **obj\_block** collision event to bring up the context-sensitive menu.
  - Place one of these markers at any position where the monster should turn around.



- Task 2 - Inheriting Behavior for **obj\_flyer**

- We have in our possession a second enemy character: **obj\_flyer**. This flyer should behave much the same as our prior monster, but it is expected that it will be able to fly up in the air without question. Still, it would be *really convenient* if we could just inherit **obj\_monster**'s code. Fortunately, for the most part we can.
- Much like Workshop 1, we can set our flyer's parent to be **obj\_monster**, and it will instantly get (inherit) all the functionality of monster. Run the game, however, and a problem immediately presents itself: all of our flyers are using monster's sprites!
- Check the **End Step** of either enemy. This is where we're setting their sprite, based on their **hspeed**. Since flyer is inheriting monster's event, the sprites being set are also monster's sprites.
- The most direct way to solve this having **obj\_flyer** override the **End Step** event (which can be done by right-clicking on the event) with a version that uses its own sprites.



Below is a quick reference for the three (3) different types of **Step** events. See the GM documentation for full details.

**Begin Step** occurs at the beginning of each step. (Disregarding create events, room start events, game start events, etc.)

**Normal Step** occurs just after alarms, keyboard events, and mouse events, and before collision events.

**End Step** occurs after collision events and after all objects are repositioned due to hspeed and vspeed.

The following challenges are optional, but completing them will further your understanding of how different facets of GML can be applied.

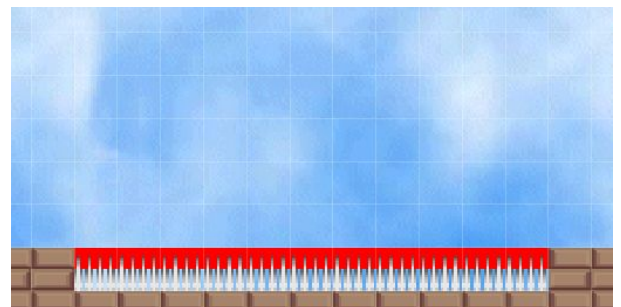
- **Challenge 1 - Using Collision-Checking to Detect Ledges**
  - Using markers is all well and good, but if your enemy characters can automatically detect when to turn around to avoid ledges, that'll save you some work in the room editor. That'll add up when your projects get bigger!
  - We've already used collision checking to detect if there's a wall in the direction a character is currently walking, and we've used it to detect if the character is/isn't standing on ground. *Try using it to detect if there's ground in the direction a character's walking.*
  - As **obj\_monster** will be moving every step, you'll want to place this check in an event that runs every step.
  - Ensure that **obj\_flyer** does not perform this check, however, as it does not care whether or not it is in the air. You'll need to make sure the event in which the check is made gets overridden. [edit] (slight wording change)
- **Challenge 2 - Using Instance Variables to Keep Track of Sprites**
  - Given monster and flyer both only have two sprites, it was fairly simple to make the change for flyer to use its own sprites. In a larger project, however, a character might have significantly more sprites unique to them, and it could be time-consuming to manually edit each individual change.
  - Do some reading about how variables can be used in GameMaker (the manual is very helpful). Variables can be assigned references to particular resources (including sprites), and the same variable in **obj\_monster** could hold a completely different reference in **obj\_flyer**. If you had 10 different places where monster needed to be assigned **spr\_monster1**, wouldn't it be great if you only had to change a reference variable *once* to assign **spr\_flyer1** instead?

- Task 3 - Squashing Enemies

- If you look in **obj\_character**, you'll see that we already have code that defeats the player upon contact with **obj\_monster** (it plays a sound and restarts the room). We're going to add to this collision event to allow the player to defeat enemies by jumping on them.
- **Let's think about the conditions required to be considered jumping on an enemy:** our character would need to be falling downward (**vspeed > 0**) and be above the enemy.
  - Coincidentally, the collision event provides the reference variable **other**, which provides access to the *other object* involved in the collision. In the case of **obj\_character**, **other** provides a reference to the enemy we collided with.
  - Comparing the two instances' vertical positions would then merely require **y** and **other.y + some offset** (e.g. 8). *Note that if using DnD actions, you should use the **If Expression** action. The syntax allowed in the Expression field is very similar to the expressions you can write in Java or C++, including relational and logical operators.*
  - There is already code that plays the player kill sound and restarts the room.
    - Add code that checks our two conditions.
    - If the jump is successful, kill the enemy instead of killing the character.
      - There is an appropriate **snd\_kill\_monster** resource you can use.
      - Replace the instance of the monster with an instance of **obj\_monster\_dead**, which is a gruesome depiction of a flattened monster. *Make sure you replace the enemy, not the player.*
      - Similar to the explosions created by the planes in Workshop 1, **obj\_monster\_dead** displays its sprite briefly, then destroys itself. **Hint: Do this using alarms.**
    - Add 50 points to the player's score.
- Fortunately, since we have **obj\_flyer** inherit **obj\_monster**, we only have to write this once.
- *Super Mario* doesn't just fall through enemies when jumping on them, he bounces up and off of them. Try setting your character's **vspeed** when successfully landing on an enemy to facilitate that bounce.

- Task 4 - Death Spikes

- There's already one way to kill the player, let's add some more.
- Check the game sprites you should find one named **spr\_death**. Similar to the marker object we used previously, this can be used to define an area that kills the player upon contact. The actual death object should be invisible, but it can be given a visual representation using the spikes present in our tileset. Use the trench on the right side of the room for this.
- At this point, we now have two ways in which the player can die.



### Recommendations:

- To avoid repeating code, such as the player's death, we should put it in a single event that can be triggered in either case.
  - Putting the code in the **Destroy** event always works, though you could also try using the **User-Defined** events as well (check the manual).
- Our game can be a very dangerous place for the character! To address this, note that we have given our character some **extra lives** to work with. This can be handled with a separate controller object or within the character itself.

### ● Task 5 - Bonus Mushrooms

- We need more ways for the player to earn points.
  - Add few mushroom objects (obj\_mushroom) to the room.
  - Implement functionality to do the following:
    - Collect mushrooms (use collisions).
    - Play a sound when a mushroom is collected.
    - Increase the score when a mushroom is collected.
  - Examine the mushroom sprite (spr\_mushroom). Note that there are many different mushroom pictures; however, all the mushrooms in our game look the same.
  - In the create event of **obj\_mushroom**, use Random Number Generation to assign each mushroom a random sprite image.
    - *Alternatively, make a single mushroom sprite with each mushroom as its own frame, then assign the sprite to a random frame.*
    - Be careful if you're using the **If Any Object At Place** DnD action or the **place\_empty()** function in GML. This method might make it difficult to walk into a powerup!
-

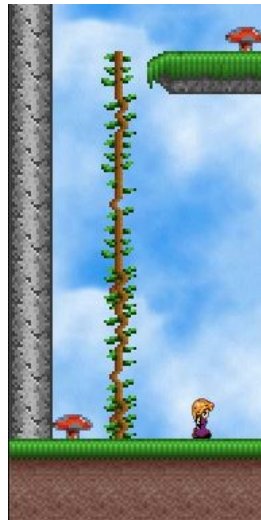
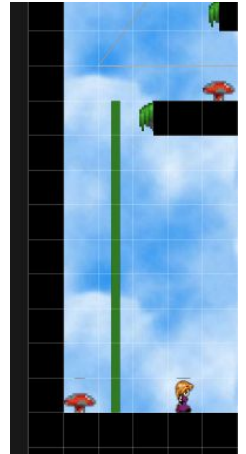
## Part 3: Extras

### Objectives:

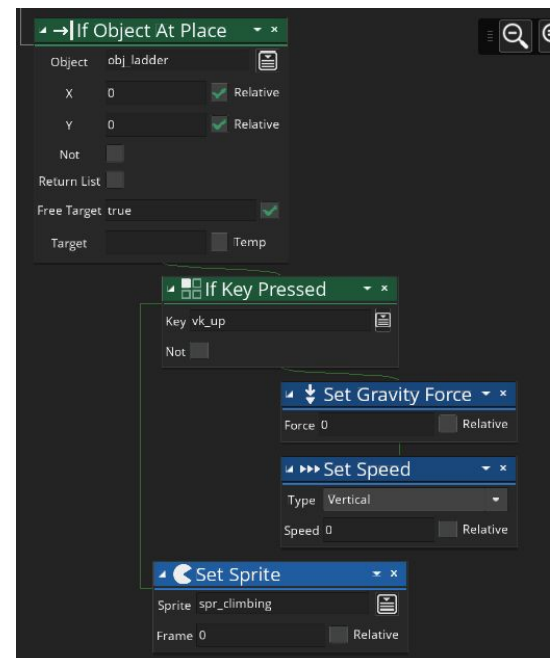
- Ladder climbing
- Multiple Levels
- Cheat Codes
- Maximizing the Screen
- Using Views
- Firing Bullets

### • Task 1 - Ladder Climbing

- Start by using the tileset to give an appropriate look to the ladder, which is the green bar on the lower-left area of the room. *This is an invisible object, so you must be editing the room to be able to see it.*
- Use the vine tiles to create a look like the one in the following image:



- Each step, our character should check if it is overlapping with a ladder. Use the **If Object At Place** action to do this.
- To initiate the act of climbing, **obj\_character** must both be on a ladder and pressing either the **up** or **down** keys
  - Recall that in GML, **keyboard\_check(vk\_up)** and **keyboard\_check(vk\_down)** can be used for this purpose.
- Note that the player character will behave very differently when climbing. **They won't be able to move left & right, and instead of jumping when up is pressed they'll climb up on the**



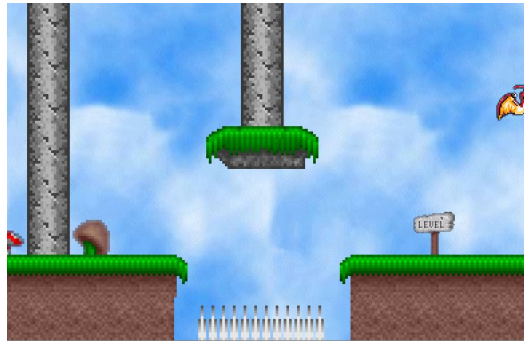


**ladder.** To enforce this drastic change in behavior, we'll need a variable that keeps track of whether or not **obj\_character** is climbing.

- For the purposes of this tutorial, we shall call the variable **climbing**. To initialize it, assign it the value **false** in the create event. This will ensure that **climbing** has a value before any other event attempts to check it.
- With our variable **climbing** declared, we need to make some edits:
  - In the condition that checks if we are pressing up or down on a ladder, set **climbing** to **true**.
  - Also set **vspeed** to 0.
  - On the other hand, set **climbing** to **false** if our character is not colliding with a ladder.
- In our left & right movement events, check that **climbing** is **false** before moving.
- Implement functionality in the up & down events similar to our left & right movements, but check that **climbing** is **true** instead. Alternatively, disable the effects of the **If Any Object At** action in the Left and Right Key Downevents.
- We should not be applying **gravity** if **climbing** is **true**.
- There is a sprite named **spr\_climbing**. Make sure that it is displayed when appropriate.

## • Task 2 - Multiple Levels

- The game resources include an object named **obj\_level\_exit** that we can use to depict the end of a level. When **obj\_character** collides with this object, the current room changes to a new one.
  - There are functions that can be used to either **Go to the Next Room** or **Go to [a] Room** of Your Choosing. Either one can work for these purposes, but it's recommended to check the manual to see how each of these work.



- In order to test this, you'll have to make an additional room (**You must have at least 2 levels total**). A good, easy way to achieve this for now is just duplicating the existing room and making some changes to it.



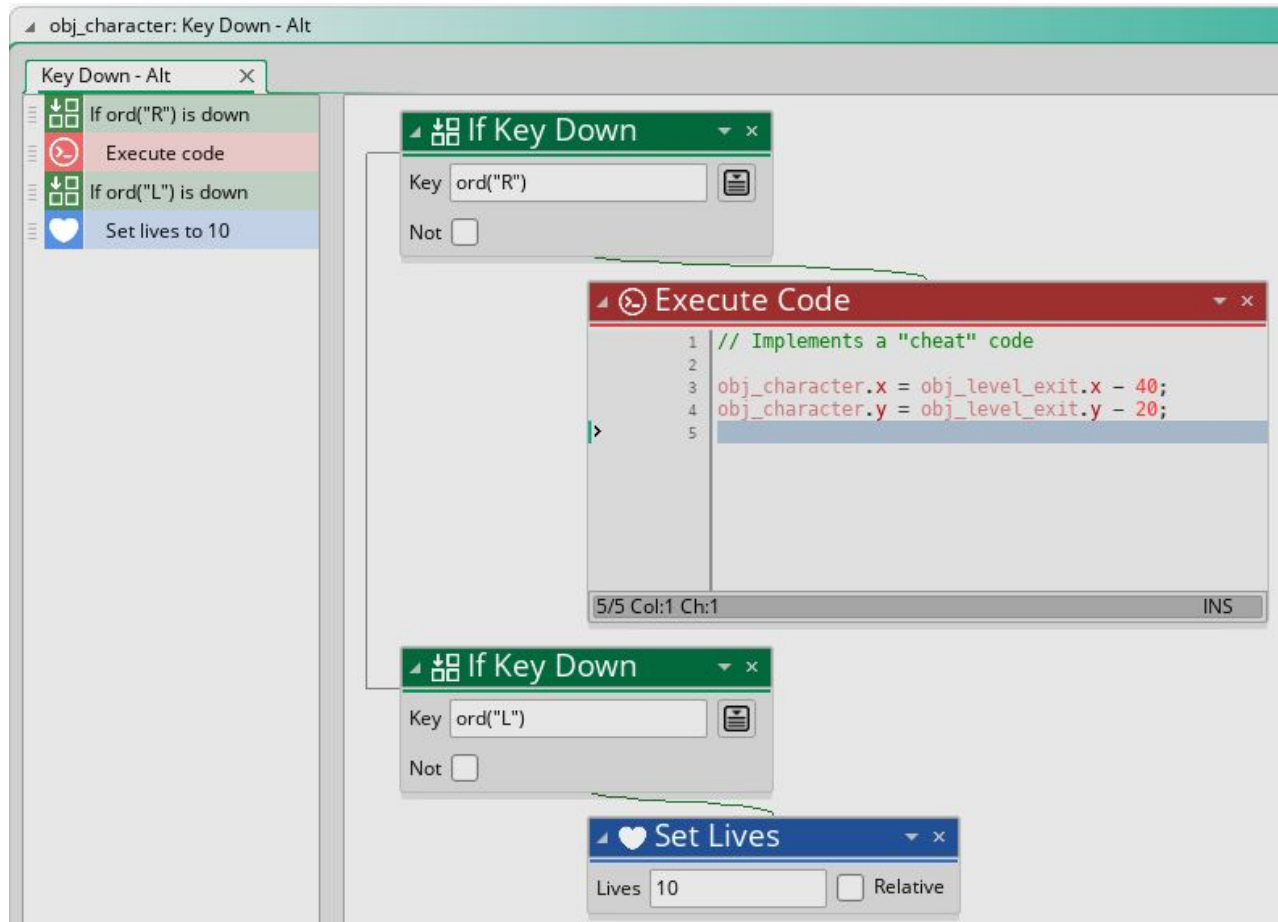
- Task 3 - Cheat Codes

- **Games should have cheat codes that facilitate testing.**

- Cheat codes perform actions such as:

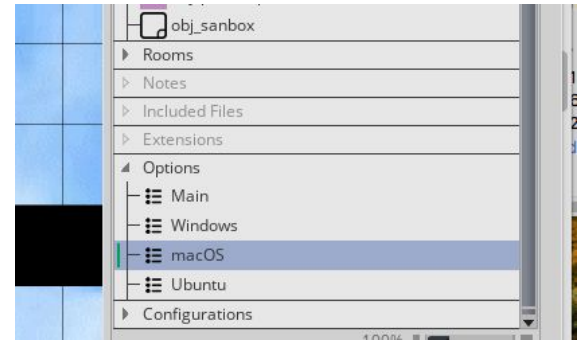
- Grant unlimited lives
    - Advance to the next level
    - Restart the game

- The following image depicts a typical way of implementing such cheat codes:



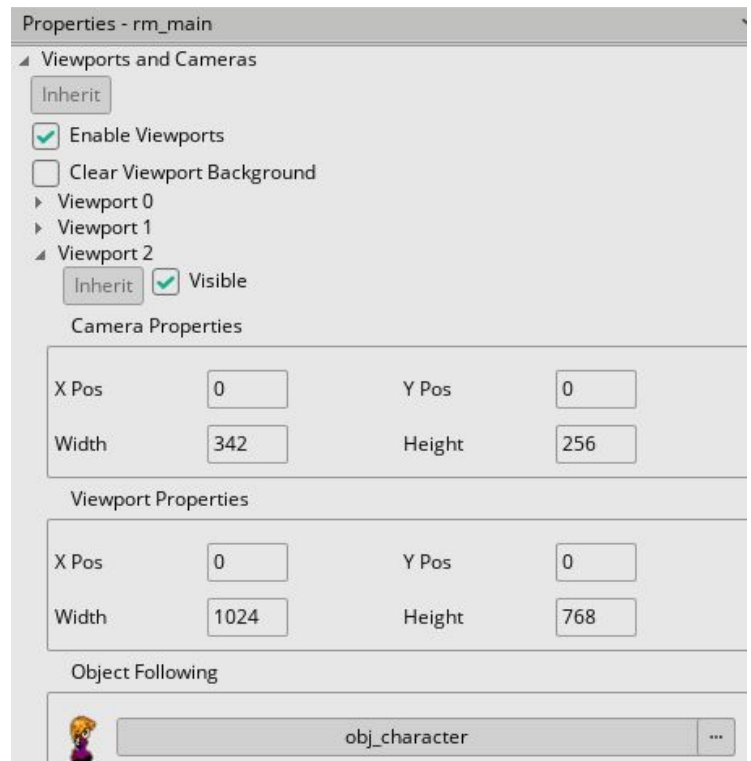
- **Task 4 - Maximizing the Screen**

- To enable full screen switching, you need to go into the **Options** dialog for each target platform. The Instructor and TA's use both Windows and MacOS, please enable fullscreen on both of these operating systems.
- Edit the object named **obj\_controller** to hold the functionality to toggle between full-screen and window mode.
- Implement two keyboard events as follows:
  - **F4 toggles fullscreen**
    - Look in the manual for a function that sets whether the game is run in fullscreen or not. If using Drag & Drop, you will need to implement this in an **Execute Code** action.
  - **F10 exits the game**

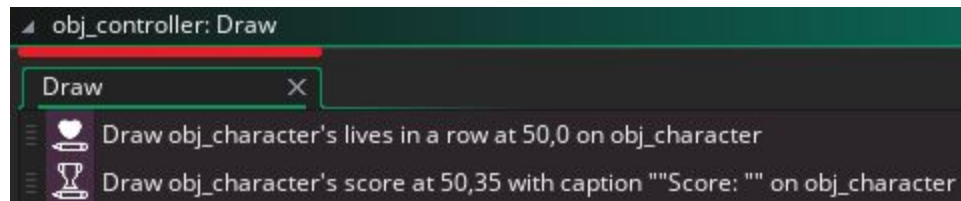


- **Task 5 - Using views**

- GameMaker Studio provides built-in functionality to enable multiple different views of the same room.
- These are configured per room, using the **Viewports and Cameras** section of the **Room Properties** pane in the **Room Editor**
- Make the game more challenging by using a viewport that restricts what the player can see.
  - Make sure that the camera area is smaller than the Viewport
  - Make sure to set object following to use the player's character.
- Create a simple mini-map, by using a camera that covers the entire screen area, but uses a small viewport.
  - *Hint: You will need to enable a different viewport*

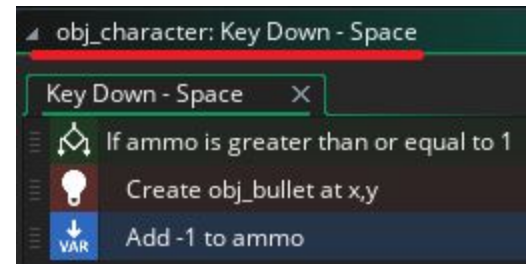


**Note: When using viewports, the x and y position will need to be fixed onto the character's x and y position.**



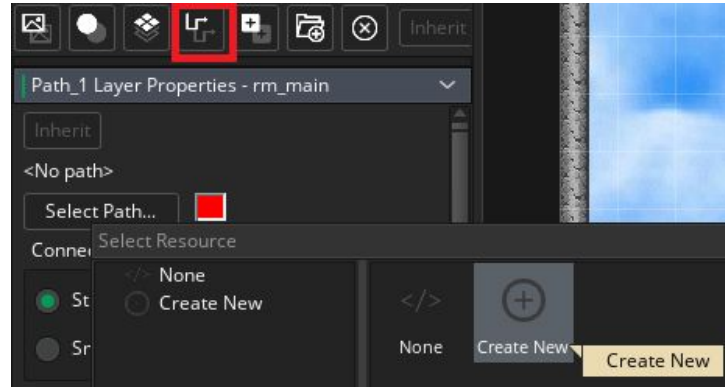
- Task 6 - Firing a bullet based on an object's sprite.

- (From this point on, assume that any new object has a corresponding sprite(s) in the resources folder for it)
- We will now add a second attack vector to our character's list of abilities. This will require ammo.
- Assign our object an **ammo** variable.
- Create an ammunition object that upon collision adds 10 to our **ammo**.
- Give our character a keyboard event that, *upon pressing space*, fires a bullet if we have **ammo** remaining.
- This **obj\_bullet** we create will have to move left or right depending on which direction our character is facing.
  - There are a number of ways to do this, one which only requires a variable that is already set in **obj\_character**.
  - Whatever method you use, the bullets should **only** move left or right, and it should be consistent with the direction **obj\_character** is facing.
- Upon collision with any enemy, **obj\_bullet** should destroy the enemy and add +10 to the game score.

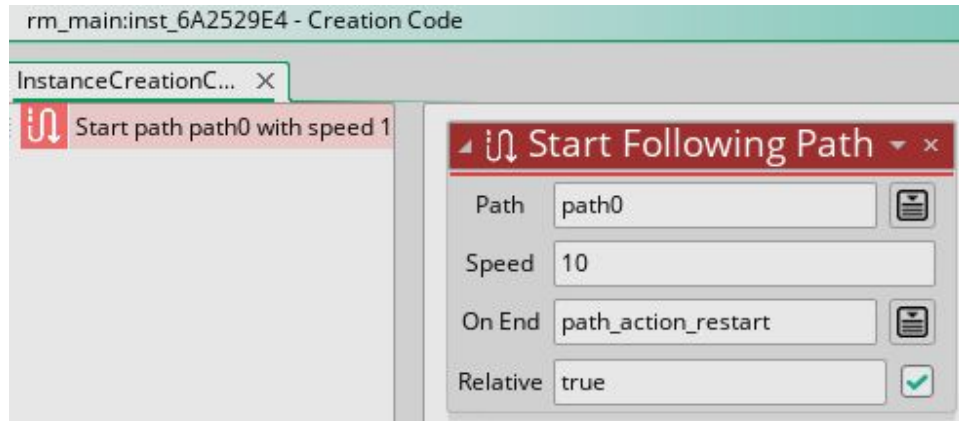


- Task 7 - Using Paths

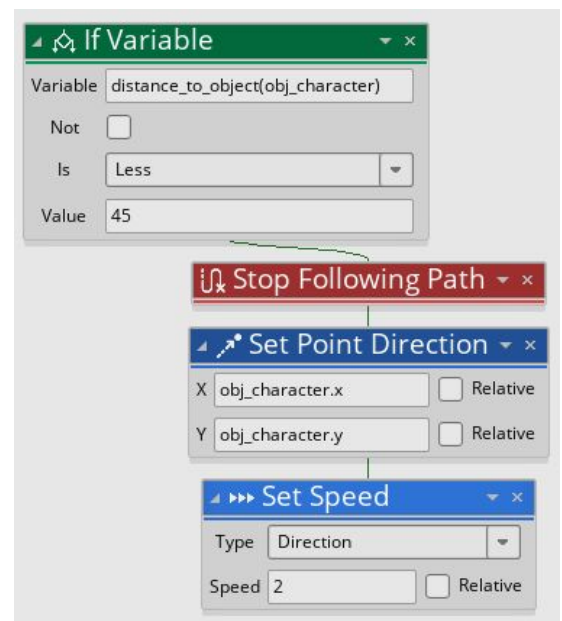
- Creating a new path layer and apply it to one of the flying monsters
  - Using the room toolbox, create a new **Path Layer** (see image)
  - Select Path and Create New path
  - You can now click on the room to create a path
  - Create a new path by selecting multiple waypoints.
    - Make sure that the path is navigable, i.e., objects should not run into obstacles and get stuck while following the path.



- Create a new object named *obj\_flyer\_with\_path*
  - On **Create** event, set the path.
  - Can be done for all instances (on object) or for a single instance in room.
  - Could also have a dedicated type of object that has a path assigned.



- Check for proximity
  - Applies to the flyer monster with path
  - **Step** event
  - Use 2 for the speed



## Additional Items NOT Required But Worth Considering:

- Middle click on a function name brings context-sensitive help
- Use of `///` comments to add JSDoc Script comments  
e.g. `/// @description My Function`
- Custom-built text boxes
- Variables can store numbers or strings
  - Use `var` for local variables, i.e., variables used only in specific events
- When drawing text, use hashtags (`#`) to create new lines
- Objects will snap to the grid when placed in a room. To change this and do exact placement, hold CTRL while dragging them.
- To change the alignment of an object with respect to the grid, use the **Flip X** and/or **Flip Y** buttons.
  - In GM 2.x object properties appear after double-clicking on the object.
- Reminder: The order of layers in the room editor determines whether a sprite is obfuscated by another. For example, drag the tile layer to the top and watch what happens when the princess character climbs.
- For sprites with multiple frames. It is necessary to set the speed to zero if only one frame will be displayed.
- Make sure to always set what draw and set DnD actions are relative to.
- Path are set in the object creation code. Double click on instance.
  - Check precision.
  - Check relative = true
  - Be careful with solids!
- When a room restarts or the player changes rooms, non-persistent objects will respawn.
- DnD Draw Value cannot be left blank.
- GameMaker Studio has an exhaustive built-in manual, which includes sample code.