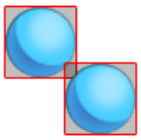
COLLISION MASKS

Although we have already worked with **Collision Events**, it if often necessary to edit a sprite's **collision mask**, which is the area that is used to calculate when two objects collide or not and trigger a collision event. One of the many useful tools that GameMaker includes is a **Precise Collision Checking** option, which performs pixel-perfect collision detection between different objects in your games. Every new sprite you create has this option enabled by default, and it means that you'll only get collision events when there is a visible overlap between the sprites of two different objects in the game.

Collision Detected

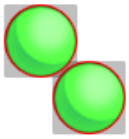


Neither of these instances have precise collisions and so they are checked based on the bounding box of the defined masks.

Collision Detected

One of these instances has a precise mask, but the other doesn't. Collisions are resolved based on the bounding box of the mask.

No Collision Detected



Both these instances have been marked as precise, so no collision is detected even though their bounding boxes overlap.

Collision Detected

These instances are in collision as they are both marked as having a precise collision mask, so they need to "touch".

PRECISE COLLISION CHECKING

It is usually better to avoid using Game Maker's **precise** collision detection option for platform games. Precise collision detection uses collision masks that are simply the visible pixels of your sprites, allowing for perfect collision detection between instances in your games. However, this can often lead to problems whereby a character or object can get stuck while walking or jumping into platforms because the collision masks are always changing to exactly match the pixels displayed in the animation frames of the character. So sometimes collisions can occur in one animation frame, but go away in the next when the sprite changes.

This can then result in the character getting stuck, flipping back and forth between sprites. There are also problems with the precise collision on the platforms themselves, where even tiny bumps stop the character from walking across them properly.

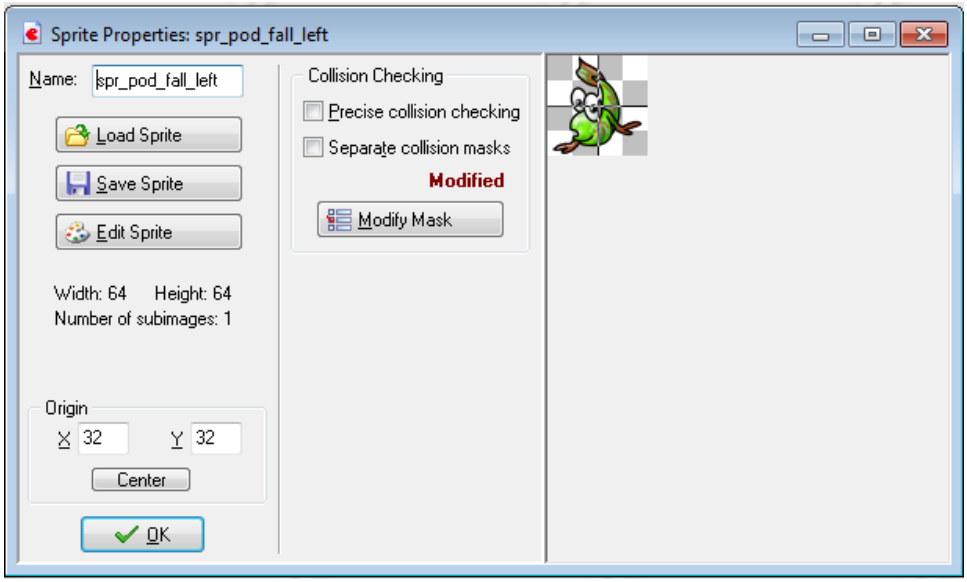
BOX COLLISION DETECTION

It is often easier to control a platform game character when the physical parts of the landscape (the character and platforms) use **bounding box** instead of precise collision. When using bounding box, you use rectangles for calculating collisions. Rectangles produce a much more solid and predictable playing experience because all the character sprites use an identically sized bounding box and there are no dips or bumps in the shapes to give unpredictable results.

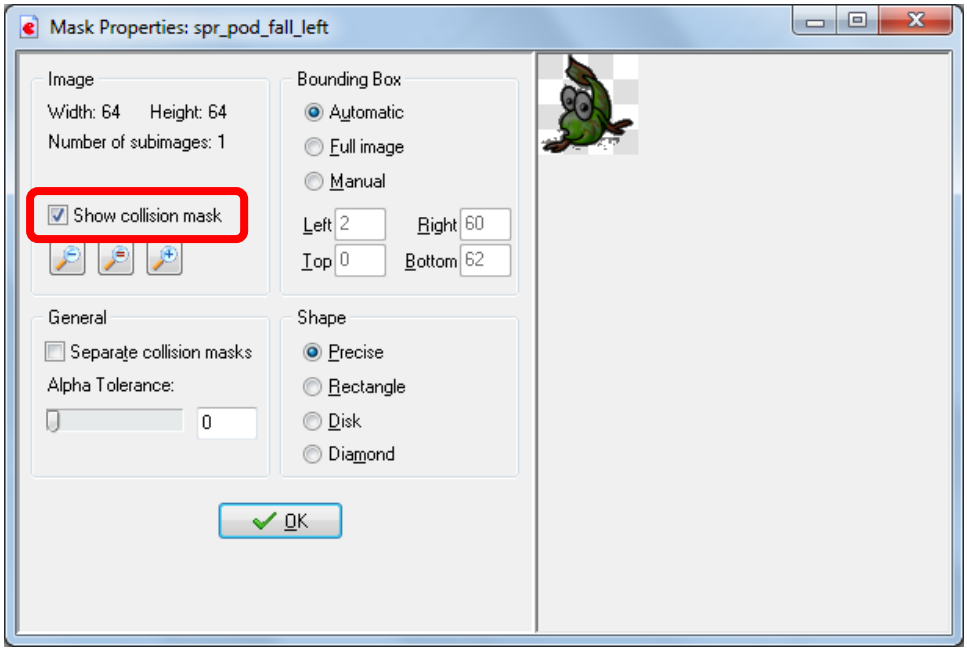
Another important point to keep in mind is that box-based collision is faster than precise collision detection because it is very quick to mathematically calculate whether two rectangles overlap, but slower to compare all the pixels between two images. Therefore, as far as collision in platform games is concerned, box-based collision is a safer, faster, and more reliable alternative to precise, pixel-based collision. That's why it's important that we make use of box-based collision detection in our platform games as well as ensuring that the dimensions of each box are the same for all the different sprites of the main character.

Let's apply this to a game called **Fishpod**. I have gone ahead and created sprites and objects for this game which you can find in the shared directory.

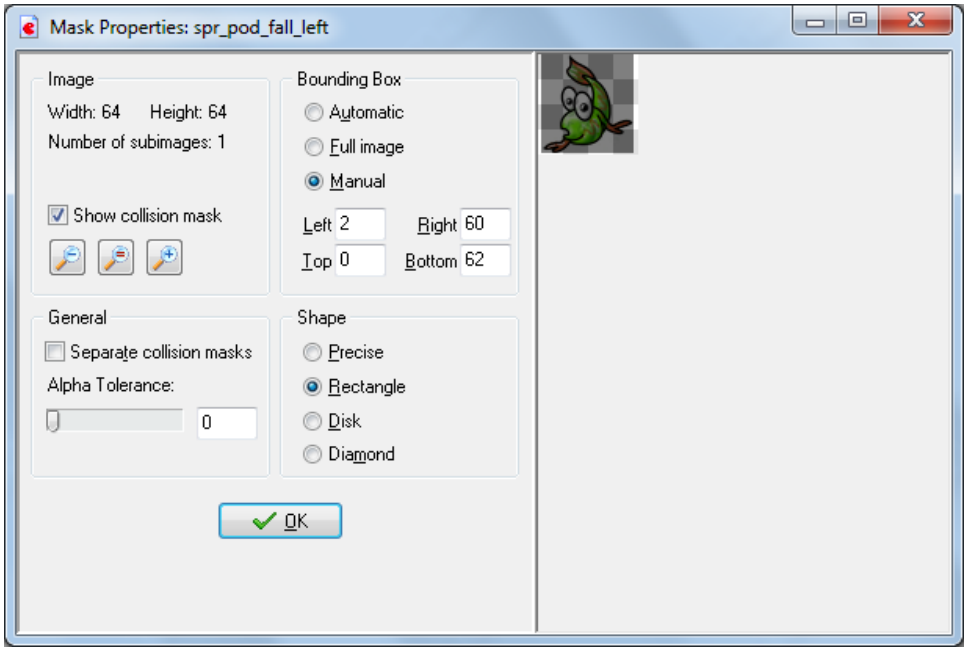
1. Open the Sprite Properties for **spr_pod_fall_left** and click the **Modify Mask** button.



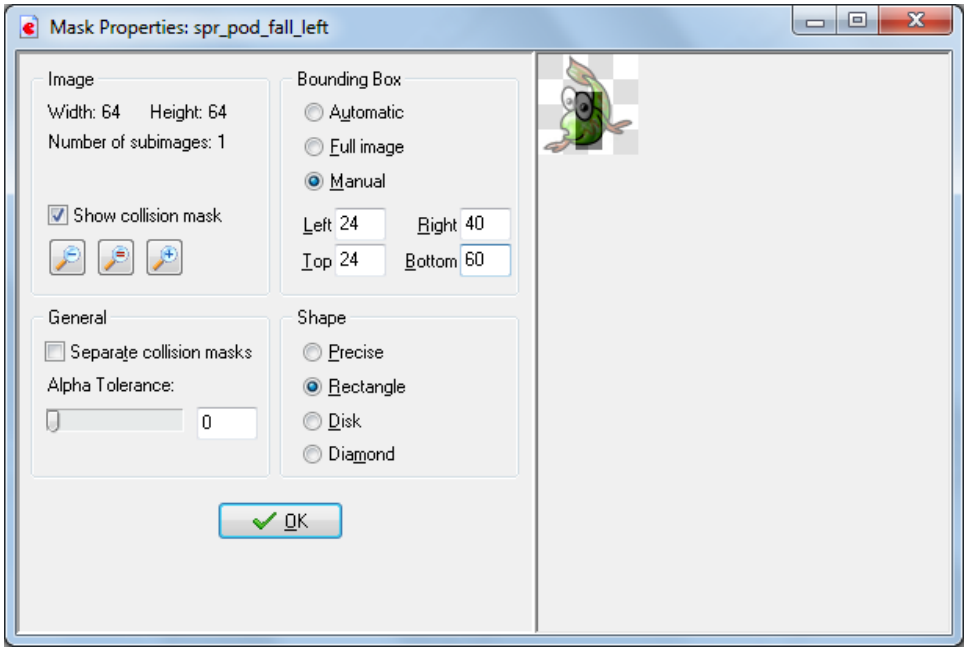
2. Under **Image**, ensure that the **Show Collision Mask** is enabled so that you can see the results of the changes you will make.



3. Under **Shape**, select **Rectangle** and under **Bounding Box**, select **Manual**.



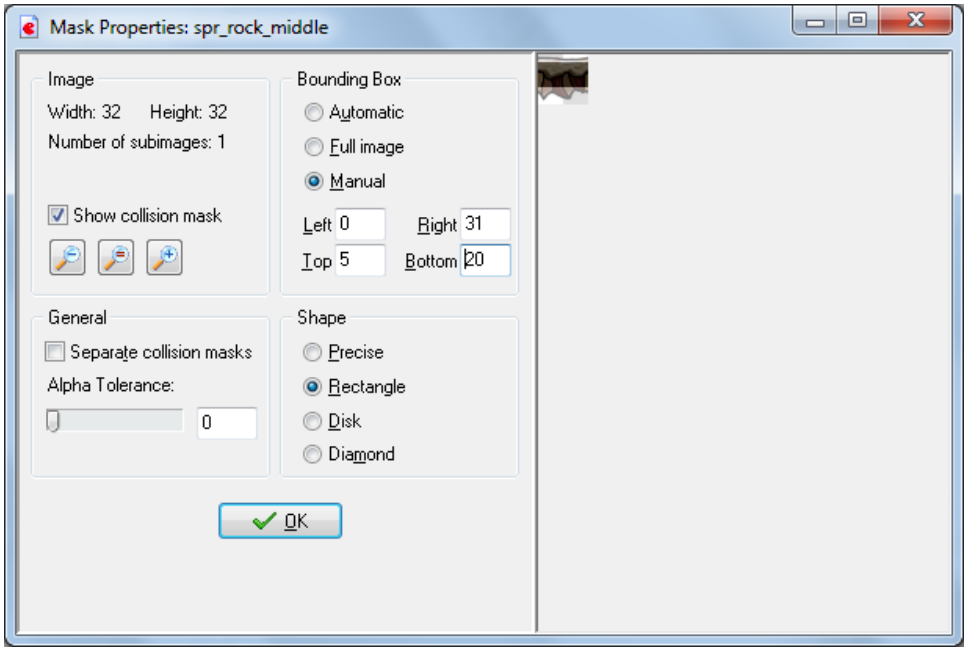
4. Set the **Bounding Box** dimensions to **Left 24**, **Right 40**, **Top 24**, and **Bottom 60**.



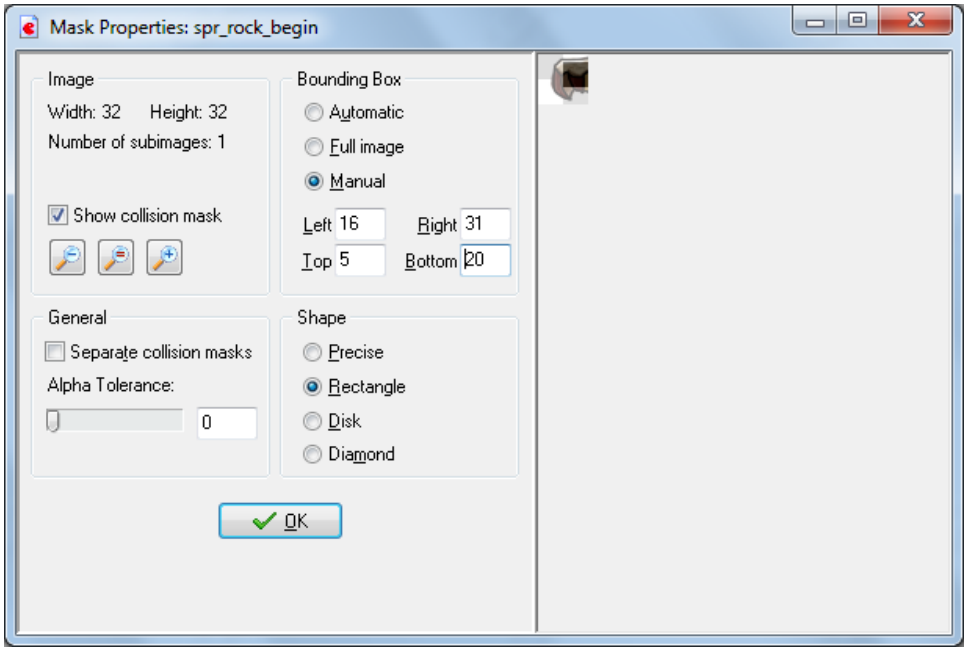
5. Repeat steps 1-4 with identical values for **spr_pod_fall_right**, **spr_pod_jump_left**, **spr_pod_jump_right**, **spr_pod_stand_left**, **spr_pod_stand_right**, **spr_pod_walk_left**, and **spr_pod_walk_right**.

   All of these sprites must have the same bounding box dimensions for the game to work correctly.

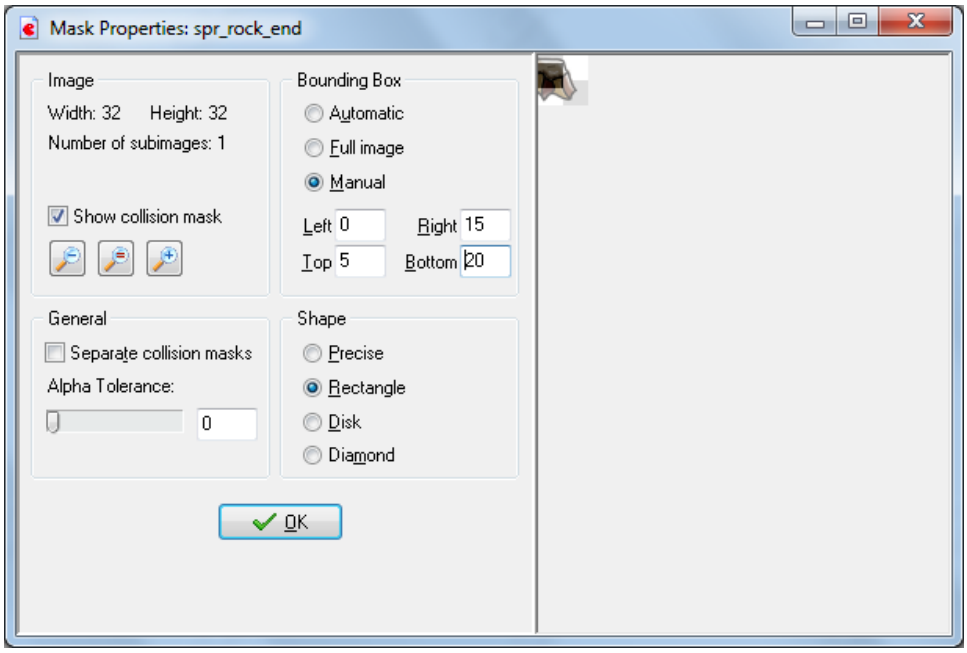
6. Repeat steps 1-4 for **spr_rock_middle** using **Left 0**, **Right 31**, **Top 5**, and **Bottom 20**. This is the full-width platform sprite. Note that the bounding box doesn't include the topmost pixels of the rock, as this allows the character to have a small overlap when it is standing on the rock. It also doesn't include much of the spiky underside of the rock as it is cosmetic.



7. Repeat steps 1-4 for **spr_rock_begin** using **Left 16**, **Right 31**, **Top 5**, and **Bottom 20**. This is a half-width platform sprite, but it still needs to have the same **Top** and **Bottom** values as the full-width platform to ensure smooth movement across them.



8. Repeat steps 1-4 for **spr_rock_end** using **Left 0**, **Right 15**, **Top 5**, and **Bottom 20**.



The remaining five sprites (**spr_lava_begin**, **spr_lava_middle**, **spr_lava_end**, **spr_gold**, and **spr_pansy**) can actually keep their default **Precise Collision Checking** setting. Fishpod won't physically interact with them in the same way because they are either hazards that will kill him instantly or collectables that will disappear on contact. Since they don't have a physical effect on Fishpod, it isn't essential for them to use the same approach to bounding-box collision.

You may also be wondering why we set up a bounding box that is much smaller than the size of the character, but there are actually two good reasons to do this:

- Firstly, it only takes a horizontal overlap of one pixel to support Fishpod standing on top of a platform. If the bounding box extended right to the end of Fishpod's flippers, then he could support himself with the very tip of his flipper in a very unrealistic way. With the smaller bounding box, he needs at least a whole flipper in contact with a platform to stand on it.

- Secondly, there is nothing more annoying in a game than when an enemy or hazard kills you when there are clearly no overlapping pixels. If we used the larger, default bounding box, then collisions could potentially occur when there was actually no actual overlap between the sprites. Using a smaller bounding box that remains within the center of the sprite helps to prevent this happening and makes for a less frustrating playing experience.

SOURCE: Habgood, Jacob, Nielsen, Nana, Rijks, Martin and Kevin Crossley. *The Game Maker's Companion: Game Development: The Journey Continues.* New York: Apress, 2010. Print.