Side-scrolling Video Games

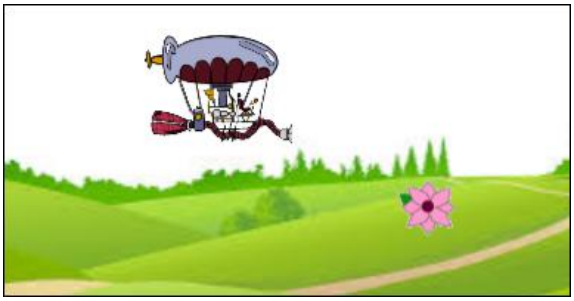
Pong was one of the earliest commercially-successful video games. Pong was an electronic ping pong game. Its success inspired other formats.

The arcade game, *Defender*, introduced a category of games in which a landscape scrolled past as the player's ship was used to defend the earth from mutants descending from the sky.

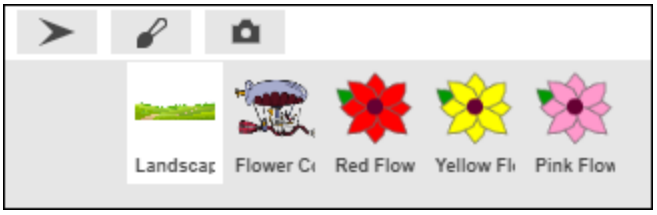
This type of game is known as a *side-scrolling* game because the landscape scrolls to the side, from right to left. The popular Mario Brothers games are another example of the side-scrolling game format.

Creating a Scrolling Landscape

A side-scrolling game format can be emulated in *Snap!* In this sample game, a blimp is used to vacuum up flowers as the landscape scrolls past. Once the basic principles are understood, the game mechanism can be adapted to create other types of side-scrolling game.

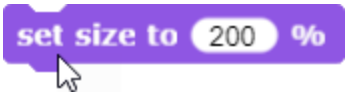


A sprite that serves as a backdrop (shown on the left-hand side in the row of sprites below) is used to create a landscape that scrolls past.



When the image of the landscape is first imported as the costume for the sprite, it is automatically scaled by *Snap!* so that it exactly fills the screen.

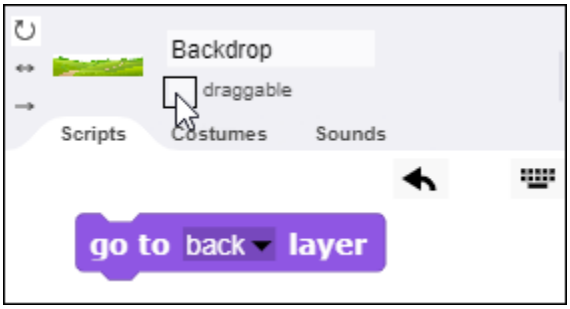
The **Set Size** code block can be used to increase the size of the sprite's costume so that it extends beyond the sides of the screen. This expands the landscape so that it can scroll across the stage.



When the size of the costume is increased, only a portion of the sprite's costume fits on the stage. The other portions of the costume extend off stage.

The code block **Set X to [X Position – 1]** can be used to move the sprite to the left by one step. If this instruction is placed in a loop, the landscape will appear to move from left to right.

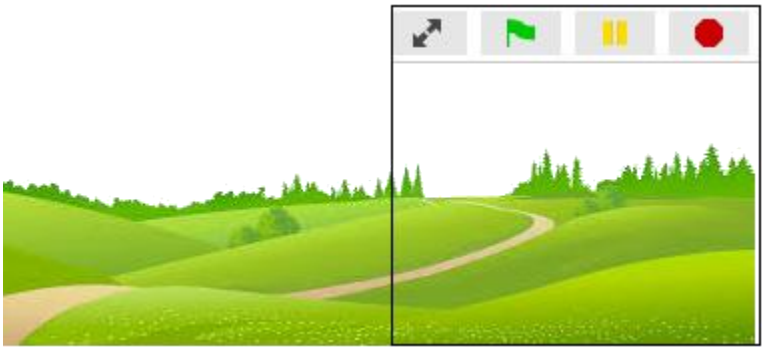
The *Draggable* option for the sprite that is the backdrop should be unchecked, so that that the player does not inadvertently drag the backdrop around the stage with the mouse.



The backdrop sprite should also be sent to the back layer of the stage so that all of the other sprites will be in front of it.

Creating a Continuously Scrolling Landscape

If the landscape is repeatedly moved to the left, eventually the right edge of the landscape will be aligned with the right edge of the screen. The default width of the stage is 240 steps wide. If the landscape is twice as wide (i.e., 480 steps wide), the right edge of the landscape will be aligned with right edge of the stage when the X position of the sprite is – 240.

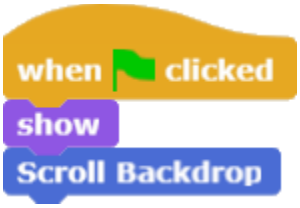


To ensure a continuously scrolling backdrop, a test can be incorporated to check when the X position of the sprite is less than "-240". At that point, the position of the backdrop should be reset to move the backdrop back to its beginning position again.

The numbers used in this example assume that the backdrop is twice the width of a stage that is 240 steps wide. The specific numbers for the X position test will depend upon the size of the backdrop.

Once the X-position numbers required are tested and verified, the code blocks can be used to create a **Scroll Backdrop** procedure that is initiated when the green *Start* flag is clicked.

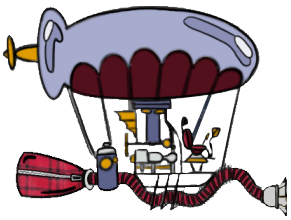
When the green flag in the upper left-hand corner of the stage is clicked, all of the code blocks attached to **Green Flag** blocks (such as the one in the illustration below) are executed.



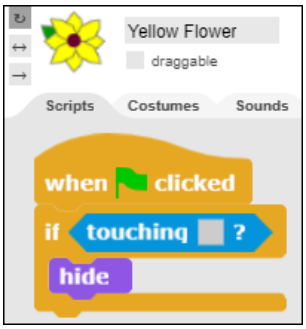
The **Green Flag** blocks make it possible to start multiple procedures across different sprites at the same time.

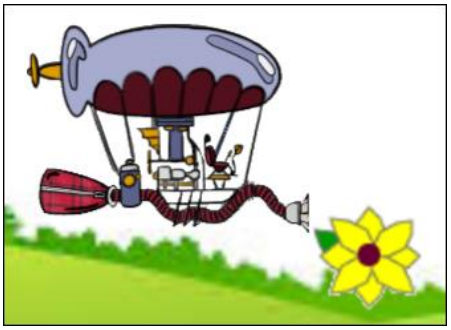
The Flower Collector

Once the scrolling backdrop procedure has been developed, a mechanism to collect flowers will be required. This vehicle consists of a blimp with a nozzle that can be used to scoop up flowers that are sucked into a collection bag at the rear of the vehicle. (The blimp is a sprite and the flowers are also sprites.)



When the nozzle of the flower collector touches a flower, the flower is scooped up. A method is therefore required to determine when the nozzle of the Flower Collector touches a flower. The nozzle of the Flower Collector is gray. The script for the flower can therefore be used to detect when the flower has been touched by the nozzle by checking to see if it has been touched by a gray object.



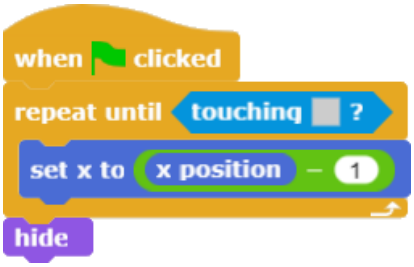


The Flower Collector sprite is moved with the mouse until its nozzle is touching a flower. At that point the mouse button is released. When the flower is touched by a gray object (i.e., the gray nozzle) and the mouse button is released, the flower disappears as it is vacuumed into the collection bag of the Flower Collector. The **Hide** code block is used to hide the flower, causing it to disappear.

If a handful of flowers with similar scripts are scattered across the landscape, a startup procedure can be used to show all of the flowers at the beginning of the game. Then each flower will disappear as it is vacuumed up until all the flowers are collected.

Flowers

The flowers need to scroll along with the landscape. The same code block that causes the backdrop to scroll, **Set X to [X Position – 1]**, can also be incorporated into the script for each flower to cause the flowers to scroll as well.

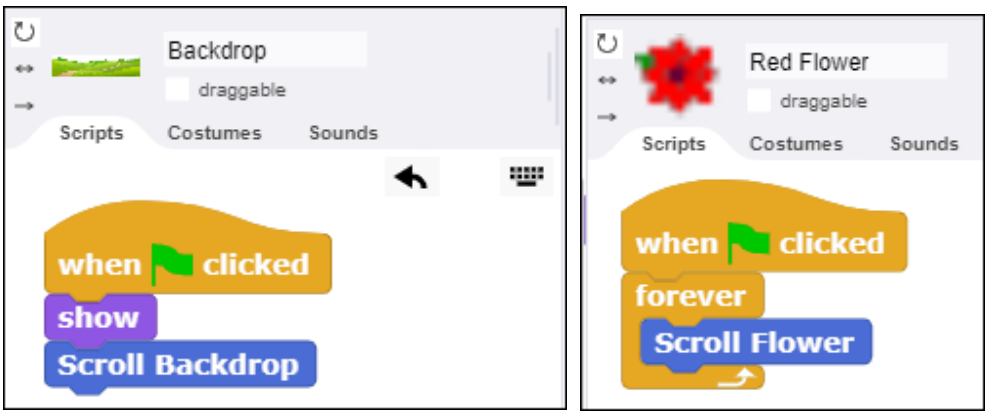


The **When I Start as a Clone** code block is used to spawn new flowers. Code blocks and procedures attached to the **When I Start as a Clone** code block are created within the new flower clones.

In this example, when a clone of a flower is created, the **Show** code block causes the new clone to be visible. The **Scroll Flower** procedure then causes the new flower to begin scrolling from left to right along with the landscape and all of the other flowers.

Distribution of Code Blocks across Clones

Each sprite can have its own code blocks. When the green flag is clicked, the code blocks attached to each code block begin to execute.

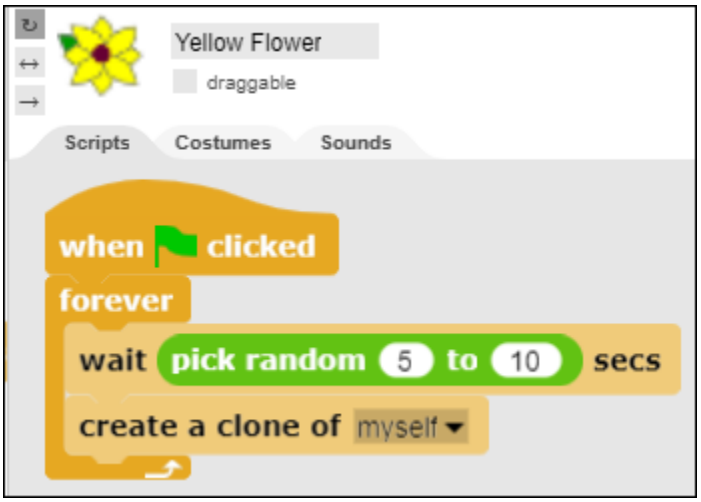


Simultaneous execution of multiple code blocks across different sprites is known as parallel processing because several sets of computer instructions are being executed at the same time (i.e., in parallel).

Spawning New Flowers

The procedures described above can be used as the framework for a complete side-scrolling game. However, the game is over when all of the flowers have been collected.

Most video games have methods of recreating new objects (flowers in this example) to enable the game to continue indefinitely. In *Snap!*, the **Clone** code block can be used to create a clone of an object. In the example below, the **Clone** instruction is used to randomly create a copy of the flower every five to ten seconds.



This ensures that new flowers are generated as previously-created flowers are scooped up by the Flower Collector.

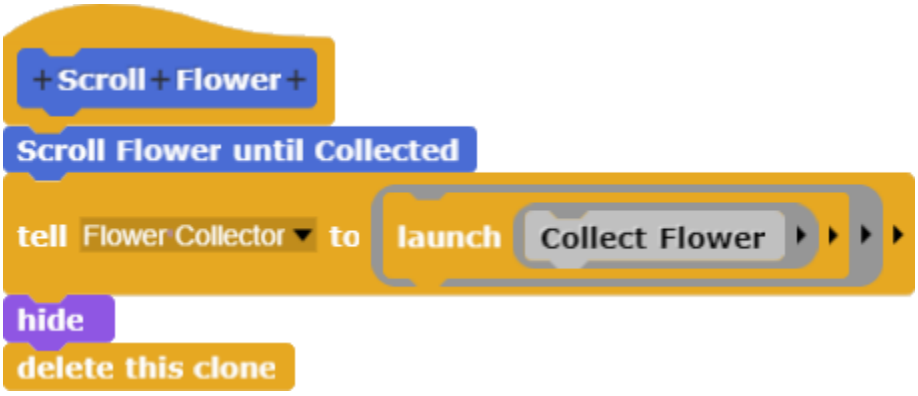
When a sprite creates a clone of itself, the clone inherits all of the scripts that were in the original sprite. The ability to create clones is a powerful capability. So that the clones do not build up, the **Scroll Flower until Collected** procedure deletes the clone if it reaches the left side of the screen without being collected.

Launching the Flower Collection Procedure

If the gray nozzle of the Flower Collector touches the flower, the flower tells the Flower Collector to launch the **Collect Flower** procedure (described in the next section).

The **Tell** code block, found under the Control code block palette, provides a way for one sprite to tell another sprite to implement an instruction.

The **Launch** code block causes the launched procedure to execute in parallel with the other scripts rather than waiting until the procedure is completed before returning control to the originating script. In this instance, use of **Launch** keeps the flower collection process from interfering with the timing involved in creation of new clones of the flower.

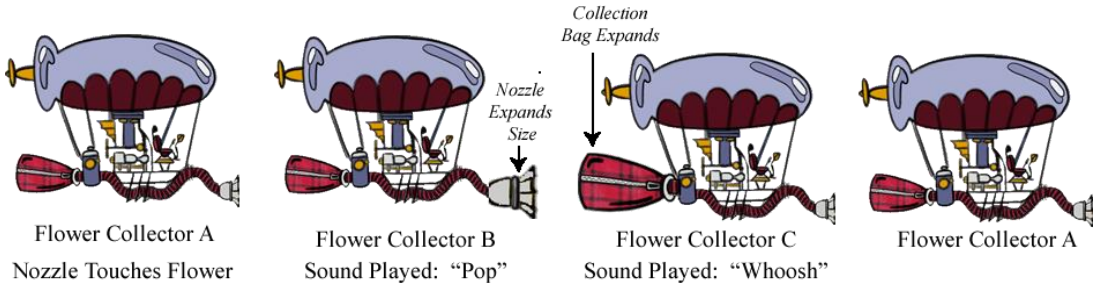


After the clone initiates the flower collection process when it is touched, there is no longer a need for the clone. Therefore, the last code block in the sequence is the instruction to **Delete This Clone**.

Adding Animation and Sounds

Sounds and animation are a significant part of a video game.

When the Flower Collector receives a message from a cloned flower to execute the **Collect Flower** procedure, it cycles through a sequence of three costumes, labeled *Flower Collector A*, *Flower Collector B*, and *Flower Collector C*. When the clone tells the Flower Collector to begin the sequence, it first shifts to the *Flower Collector B* costume. The nozzle size is expanded in this costume. A popping sound is also played as the flower is plucked.



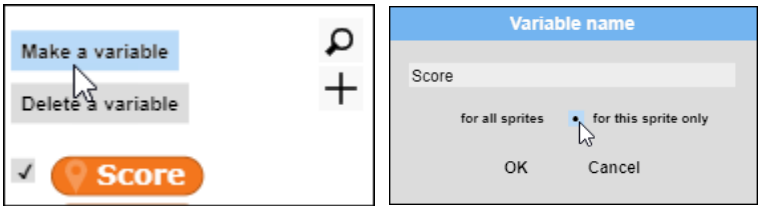
The Flower Collector then shifts to the *Flower Collector B* costume. The collection bag expands in this costume, as the flower (presumably) enters the bag. A whooshing sound is also played as the flower is sucked into the bag. The Flower Collector then reverts to its original costume.

The code blocks used to implement these steps in the **Collect Flower** procedure are shown below.

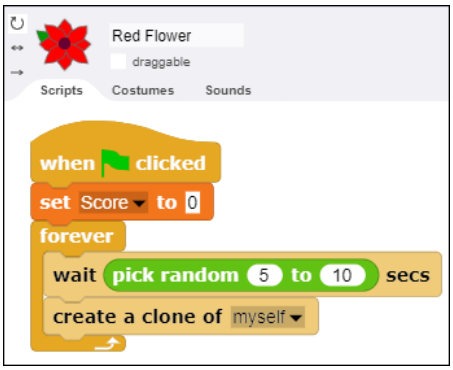
Within the **Collect Flower** procedure, the switch to the *Flower Collector C* costume until the sound "Pop" is completed. However, because the clone script used the **Launch** code block to launch the **Collect Flower** procedure, playing the sound does not hold up or impede execution of any other scripts that take place outside of the **Collect Flower** procedure.

Scoring

Most games have scores. To add a scoring mechanism to the game, create a variable named *Score* for each flower. When each *Score* variable is created, select *For This Sprite Only* as an option. This will make it possible to track how many of each type of flower have been collected.

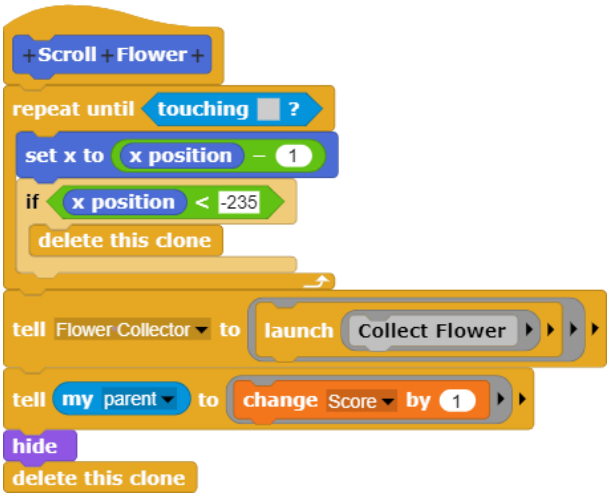


The *Score* variable for each flower is initialized (i.e., set to 0) when the green flag is clicked.



When the nozzle of the Flower Collector touches the flower and the flower is collected, an additional code block tells the parent flower to increase the *Score* variable by 1. (The **My Parent** code block is found in the *Sensing* code palette.)

This code block is added to the **Scroll Flower** procedure.



This addition increases the value of the Score variable by 1 each time a flower is collected.

There are many other variations that could be used to enhance the score process. The flowers could be assigned different point values that are tracked as the flowers are collected. Mixing in weeds among the flowers would provide opportunities to lose points as well as gain them.

The scroll speed could be increased as the player progresses through various levels to make the game more challenging. This would enable players to attempt to attain as many levels as possible.