

5. Animated Art

Glen Bull, Jo Watts, and Rachel Gibson

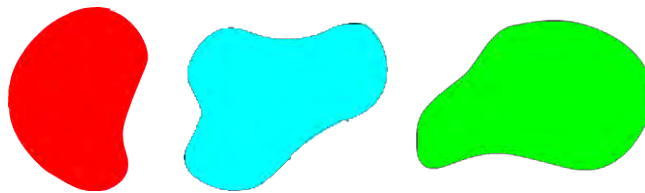
Disney's film, *Fantasia*, released in theaters in 1940, explored animated art. The renowned orchestra conductor, Leopold Stokowski, collaborated with Disney on development of themes based on "instrumental coloring."



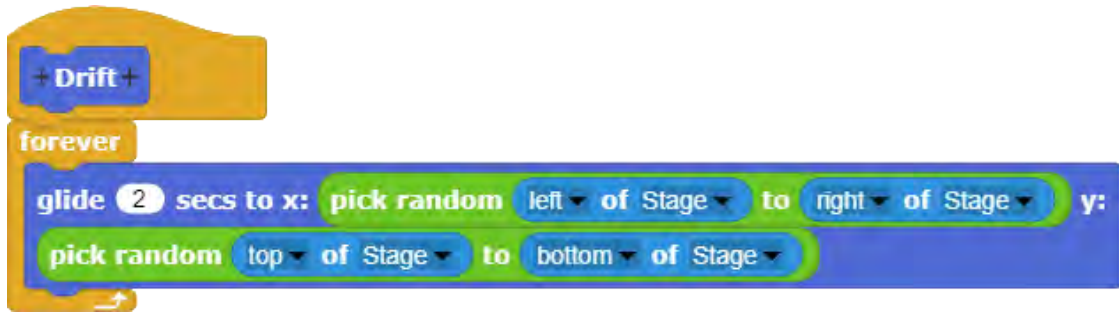
The film was recognized with a special Oscar at the Academy Awards for creating a new form of music visualization, thereby extending use of film and animation as an art form. Disney subsequently collaborated with a number of artists, including the surrealist Salvador Dali, on explorations in animated art.

Topic 5.1 Animating Objects

The programming language Snap! lends itself to exploration of abstract animated art. In the exploration that follows, a mosaic of shifting shapes and colors will be animated. This animation begins with selection of several shapes. The following shapes were created for the animation. Each shape was assigned to a different sprite.



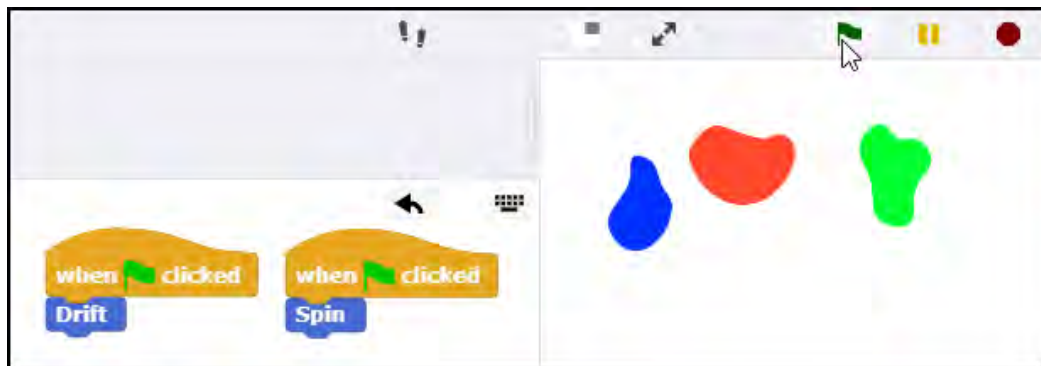
A Drift procedure causes each shape to drift across the screen. The Drift code block picks a random point within the bounds of the stage, and causes the sprite to glide toward the specified point.



A parallel Spin procedure rotates the shape as it drifts across the stage.



If each of these blocks is attached to a Green Flag block, clicking the green flag in the upper-right hand corner of the stage will cause both procedures to run in parallel.



If these procedures are created in the script space for the sprites associated with each shape, clicking the green flag will cause each of the shapes to spin as they drift across the screen.

Computing Concepts

Procedures can run either sequentially or parallel to one another. Sequential procedures complete each block of code they contain in order and one at a time. Parallel procedures complete multiple blocks of code simultaneously to accomplish tasks that would be impossible with only sequential code. In this section, the Green Flag block runs the drift and spin procedures in parallel, while the code within those procedures runs sequentially.

Exploration 5.1 Animating Objects

Create shapes for each of three sprites. Develop procedures that cause the sprites to turn and move as they drift across the screen.

Topic 5.2 Exploring Red-Green-Blue (RGB) Color

Shifting the colors of the shapes as they drift across the screen emulates another element of Fantasia’s “instrumental coloring.” Each shape on the computer screen is constructed from tiny dots known as picture elements or *pixels*. The illustration below shows an image composed of four pixels magnified ten times. An image consists of a mosaic of pixels. This image consists of one black pixel, one red pixel, one green pixel, and one blue pixel.



The color of each pixel is controlled by values (ranging from 0 to 255) that specify the amount of red, green, and blue in the picture element. This method of specifying color is known as Red-Blue-Green (RGB) color. A custom code block, Set Color, can be used to set the sprite’s current costume to the red, green, and blue values supplied as an input to the procedure. In the illustration below, the values of 128 (Red), 255 (Green), and 255 (Blue) supplied as inputs to the procedure result in a cyan color.



Details of construction of the Set Color block are provided in the *Appendix*.

Computing Concepts

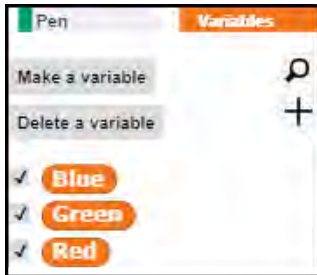
Computers can display graphics and play sound. Data representation refers to the way in which graphics, sound, and other artifacts are represented by internal numbers inside the computer. In this instance, the value of each picture element (pixel) is controlled by the amount of red, green, and blue in the pixel. This is specified by three numbers. If red and green are 0 and blue is 255 (the maximum value) then the shape will be completely blue. In the illustration above, if green and blue are the maximum value and red is half the maximum value, an aqua shade results.

Exploration 5.2 Exploring Red-Green Blue (RGB) Color

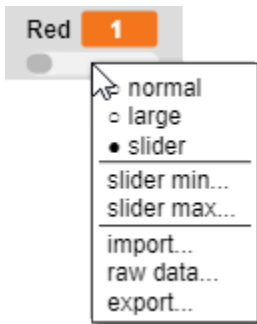
Use the Set Color code block to change the color of a sprite’s costume.

Topic 5.3 Adjusting Colors

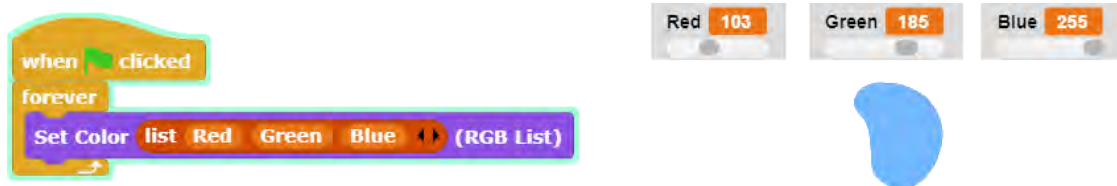
The colors obtained with varying combinations of red, green, and blue can be explored through creation of variables for each of these values. When the checkbox beside each variable is checked, a display called a *watcher* will show the current value of the variable on the stage.



Right-clicking the on-screen variable watcher accesses an option to create a slider control for the variable and assign minimum and maximum values to it.



When the Set Color code block is placed in a Forever loop with these variables assigned as inputs, adjusting the sliders controls the respective red, green, and blue values of the sprite's current costume.



This method can be used to create an Adjust Color code block that can be used to quickly determine the relative amounts of red, green, and blue that must be used to achieve a desired color.



Computing Concepts

Since the table of numbers representing the pixels in a shape are not easy for a human to interpret, a set of sliders controlling the relative red, green, and blue values makes it possible to explore the way in which mixing the colors together affects the shape.

Exploration 5.3 Adjusting Color

Create a set of sliders that can adjust the red, green, and blue shades in a sprite's costume. Identify a family of several colors that go well together, and record the RGB values for each of these colors.

Topic 5.4 Varying Color and Movement

Once the appropriate color ranges have been selected, the Set Color code block can be used to pick random colors for each of the shapes as they drift across the screen.



Fading the shape from one color to another as it moves across the screen is another method of varying color. This can be achieved by holding two of the RGB values constant while using a loop to vary the third color input. For example, if the red and green inputs are held constant (with *Red* set to a maximum value of 255 and *Blue* set to the minimum value of 1), varying the *Green* input from 1 to 255 will cause the shape's color to fade from red to yellow.



Exploration 5.4 Varying Color and Movement

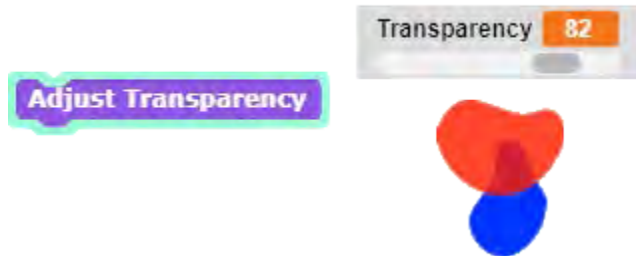
Create a block that fades a sprite's costume from one color to another as it moves across the screen.

Topic 5.5 Transparency

A custom Set Transparency code block can be used to set an image to a desired level of transparency. The underlying details of this code block are described in the *Appendix*.

Set 50 Transparency (1 to 100%)

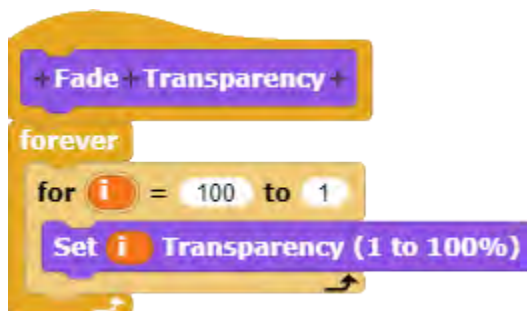
In the same way that sliders were created to adjust the red, green, and blue values of the pixels, a variable can be created with an associated slider to adjust the transparency of an image.



The Adjust Transparency code block consists of a Forever loop that continually sets the transparency value of the image's pixels to the value set by the *Transparency* slider.



In the same way that a procedure can be created to cause the shape to fade from one color to another (for example, from red to yellow) as it moves across the screen, a procedure can be created to cause the shape to fade from opaque to transparent as drifts across the screen.



Exploration 5.5 Varying Color and Movement

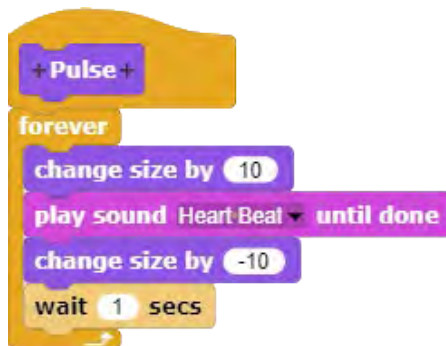
Create a block that fades the transparency of sprite's costume from opaque to transparent as it moves across the screen.

Topic 5.6 Pulsing Shapes

Shapes can be created that appear to pulse by momentarily increasing the size of the shape and then decreasing the size a moment later.



An extension of this procedure can play a sound each time that the shape pulses.



Until now, the code blocks for each sprite have been placed in the script area for that sprite. However, it is also possible to create a single custom code block that issues commands to several sprites.



One of the advantages of this approach is that revising the code only involves editing the code in a single location rather than in several different places.

Computing Concepts

In the first section of this chapter, the Green Flag block was used to launch parallel procedures. In this section, shapes are made to pulse in a procedure that will be incorporated as part of a larger program. Since the procedure will be used in a program that's already running, the Green Flag

block cannot be used here. Instead, the Tell block alerts different sprites that it's time to execute their procedures in parallel with one another. These sprites will continue to run their procedures in parallel while the overall program continues to run its statements and procedures in series.

Exploration 5.6 Pulsing Shapes

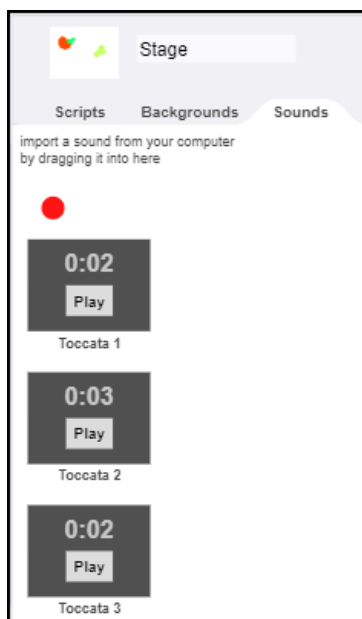
Create a block that causes a sprite's costume to pulse as it moves across the screen.

Topic 5.7 Pulsing Sounds

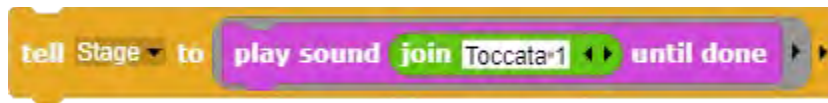
At the beginning of *Fantasia*, artists and musicians collaborated to play Bach's Toccata and Fugue in D Minor synchronized with pulsing shapes that moved across the screen.



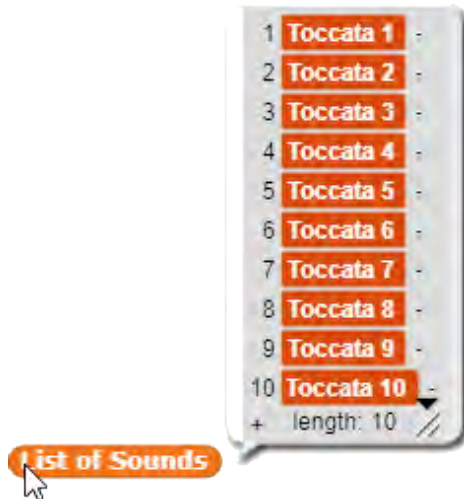
To create a similar effect in Snap!, a series of musical notes and phrases were imported into Snap!. In this instance, the musical phrases were imported into the sound files associated with the Stage. The first phrase was named "Toccata 1", the second phrase in the musical sequence was named "Toccata 2", etc.



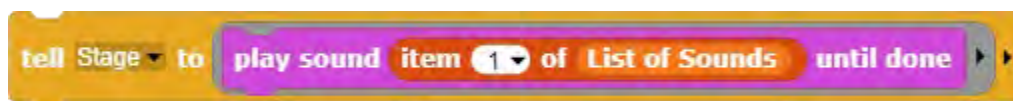
Any of the imported sounds can be replayed by telling the Stage to play the desired sound; in the example below, the Stage has been asked to play the sound named “Toccata 1”.



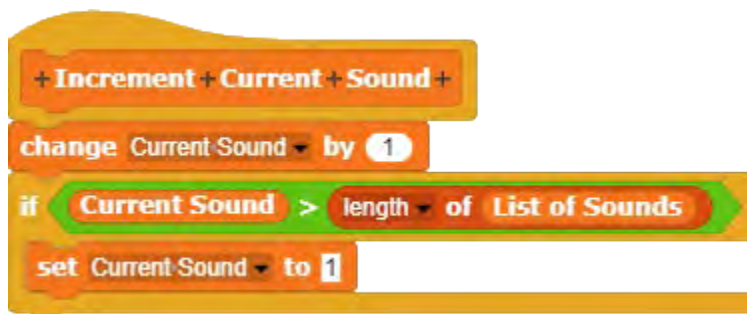
In order to play the sounds in order as the shapes pulse sequentially (i.e., one after another), a variable named *List of Sounds* containing the names of each of the sounds was created.



Telling the Stage to play the sound that is Item 1 in the *List of Sounds* has the same effect as telling the Stage to play the sound named “Toccata 1”.



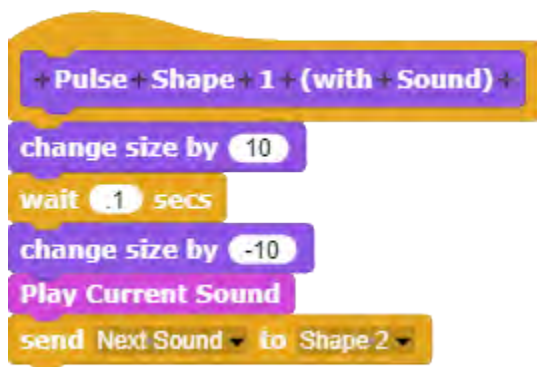
In order to move through the list while keeping track of the current sound, a second variable named *Current Sound* was created. This variable is initially set to 1; each time a sound is played, the variable is incremented (i.e., increased by 1).



Once a method for incrementing the current sound has been developed, a Play Current Sound code block can be created that plays the current sound and then increments the current sound.



When the *Green Flag* is clicked to start the program, a Pulse Shape 1 procedure pulses the shape and plays the current sound. It then sends a message (*Next Sound*) to Shape 2 asking it to play the next sound.



When *Shape 2* receives this message, the Pulse Shape 2 code block pulses the shape, plays the current sound, and sends a *Next Sound* message to *Shape 3*.



After repeating the process, *Shape 3* sends a *Next Sound* message to *Shape 1*, completing the loop.

Computing Concepts

Relational operators compare the values of two inputs and then report whether the statement expressed is true or not. Relational operators contain the symbols =, ≠, <, >, ≤, and ≥. For example, if the integers 4 and 3 were put into the = operator such that the statement read 4=3, the operator would output the value “false”. If the statement instead read 4=4, the operator would output the value “true”.

Relational operators are often used in conditional statements. Conditional statements change the order in which procedures in a program run, depending on whether or not set conditions are met. For example, given the statement “if it’s raining, take an umbrella”, you would know when you’re supposed to take an umbrella. You would also know that if it’s not raining, an umbrella may not be needed. The same is true of computer code. When a procedure sees a conditional IF block, it will check to see if the conditions described in the operators section of the block have been met. If the conditions have been met, it will execute the code within the block before moving to the next

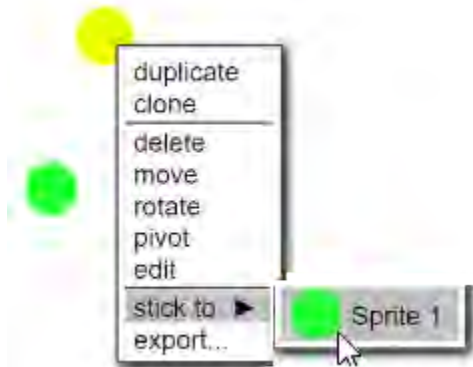
statement. If the conditions have not been met, it will skip the code in the block and continue with the rest of the procedure.

Exploration 5.7 Pulsing Sounds

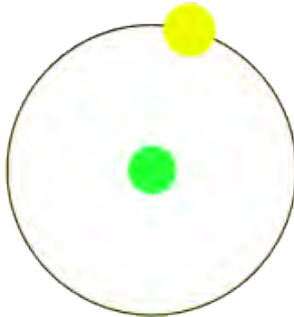
Create a list of sounds. Develop a code block that causes each sprint in turn to play the next note in the list.

Topic 5.8 Rotating Shapes

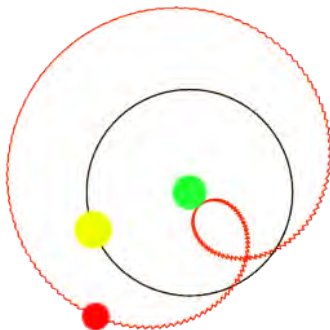
Thus far the movement of each sprite has been independent of the other sprites. Interesting movements can be generated by linking sprites together.



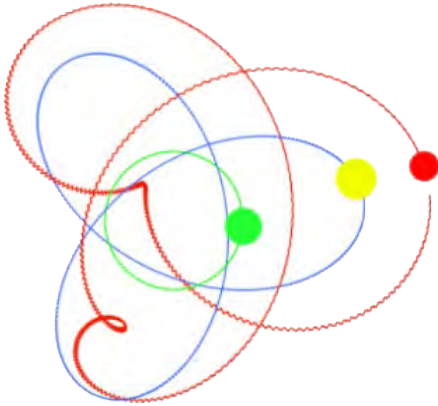
For example, if the yellow sprite in the illustration (Sprite 2) is linked to the green sprite (Sprite 1), rotating the green sprite will cause the yellow sprite to draw a circle if its pen is down.



If a red sprite is linked to the yellow sprite, rotating the yellow sprite causes the red sprite to draw a cardioid pattern.



If the green sprite moves in a circle at the same time, an even more intricate pattern is generated.



In ancient times it was understood that if the vertical distance from a point on a circle to the midline of a circle is graphed as the point move around the circle, the figure today known as a sine wave results. If the oscillation of a sinewave is played through a computer speaker, a pure tone results. Synthesized music can be created by combining tones generated in this way.

This connection between circular motion and sound was understood long before computers. The sixteenth century astronomer, Kepler, speculated that the movement of the planets around the sun might generate overtones known as harmonics. Later the composer Holst was inspired to compose the orchestral work known as the Planets that celebrated the “music of the spheres.” Therefore, this might be a fitting accompaniment to the movement generated by linked sprites.

Exploration 5.8 Rotating Shapes

Use linked sprites to design a graphic.

Appendix A

Code Blocks Introduced

A. Motion Palette

Glide

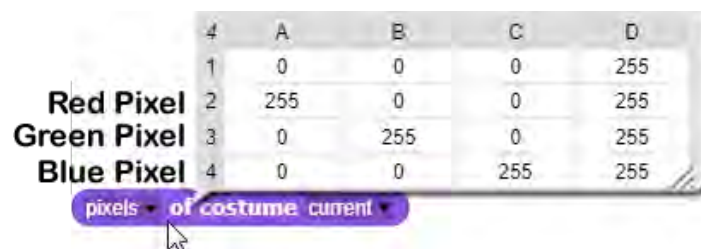
The Glide block causes the turtle to move smoothly to a specified position over a specified period of time. t



B. Looks Palette

Pixels of Costume Current

The Pixels of Costume Current code block displays the Red, Green, and Blue (RGB) values for each pixel. In the graphic below, each of the four rows represents one of the four pixels in the image.



The image shows a code block labeled "pixels of costume current" with a mouse cursor pointing to it. To the right of the code block is a table with 4 rows and 5 columns. The first column contains row numbers 1 through 4. The next four columns are labeled A, B, C, and D. The table contains the following data:

	A	B	C	D
1	0	0	0	255
2	255	0	0	255
3	0	255	0	255
4	0	0	255	255

Labels to the left of the table indicate the color for each row: "Red Pixel" for row 1, "Green Pixel" for row 2, and "Blue Pixel" for row 3. Row 4 is unlabeled.

This code block can be combined with the Switch Costume code block to control the colors of the pixels in a costume. (See Appendix B.)

Change Size

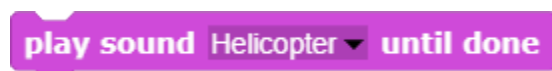
The Change Size code block is used to change the size of the turtle.



Sound Palette

Play Sound Until Done

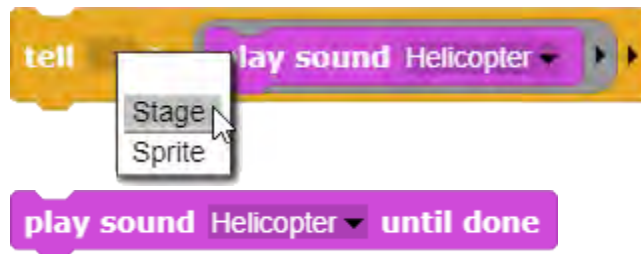
The Play Sound until Done block plays the selected sound and waits until the sound is done.



Control Palette

Tell Code Block

The Tell block tells a sprite or the stage to perform a specified action. This enables one actor such as a sprite to tell another entity such as the stage to perform an action.

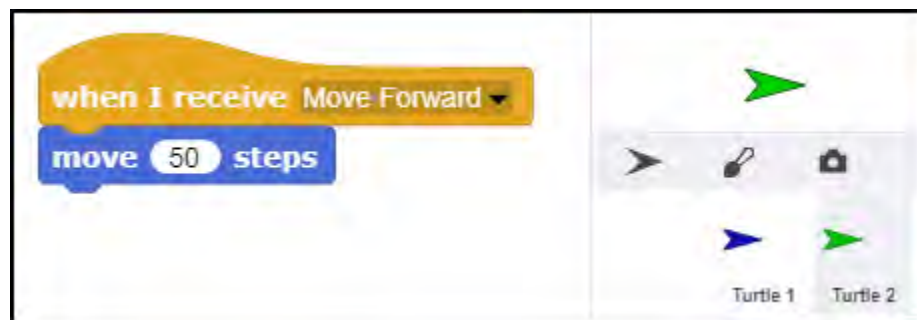


Send and Receive Message

The Send [Message] block enables one sprite (such as Turtle 1 in the illustration below) to send a message to a second sprite (Turtle 2 in this example).



When the second sprite (Turtle 2) receives the message, it performs the specified action (Move 50 Steps in this instance).



Appendix B

Adjusting the Color of a Sprite's Costume

Picture Elements (Pixels)

Each shape on the computer screen is constructed from tiny dots known as picture elements or *pixels*. The illustration below shows an image composed of four pixels magnified ten times. An image consists of a mosaic of pixels. This image consists of a one black pixel, one red pixel, one green pixel, and one blue pixel.



The color of each pixel is controlled by values (ranging from 0 to 255) that specify the amount of red, green, and blue in picture element. This method of specifying color is known as Red-Blue-Green (RGB) color.

RGB Colors

The Pixels of Costume Current code block displays the RGB values for each pixel. In the graphic below, each of the four rows represents one of the four pixels in the image.

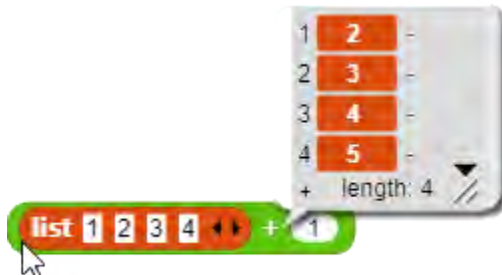
	4	A	B	C	D
	1	0	0	0	255
Red Pixel	2	255	0	0	255
Green Pixel	3	0	255	0	255
Blue Pixel	4	0	0	255	255

pixels of costume current

The first three columns (A, B, and C) control the red, green, and blue values of each pixel. The fourth column (D) controls the transparency of the pixel, ranging from completely opaque (255) to completely transparent (0). Because the first pixel, represented by Row 1, has values of 0 in each column, it is completely black (i.e., has no color). The second row has a maximum value of 255 in the first column (i.e., the red column) and 0 in the green and blue columns. Therefore, this pixel is completely red.

Changing the Colors of Pixels

The Switch to Costume code block can be used to change the values of pixels in the current costume. All of the elements in a list can be changed by arithmetic operators. For example, if one is added to a list of numbers from one through four, a second list of the numbers two through five results.



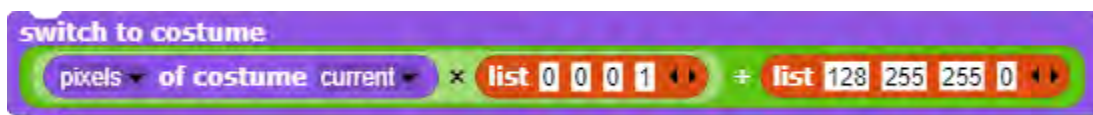
Therefore, if the first three columns of values for the picture elements are multiplied by zero, the red, green, and blue values for those pixels are set to zero. (Multiplying the fourth column controlling transparency by one leaves its values unchanged.)



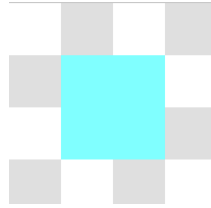
Once multiplication is used to reset the color values to zero, the addition operator can be used to add the desired amounts of red, green, and blue color. (Adding zero to the fourth column controlling transparency leaves its values unchanged.)



The Switch to Costume code block can then be used to switch the pixels to the desired color values.



Values of 128 (Red), 255 (Green), and 255 (Blue) result in a cyan color.



This method of changing the value of a table of numbers is known as a *matrix operation*. Fortunately, the complexity of the underlying matrix operations involved in adjusting the color of a sprite's costume can be encapsulated in the form of a custom code block.



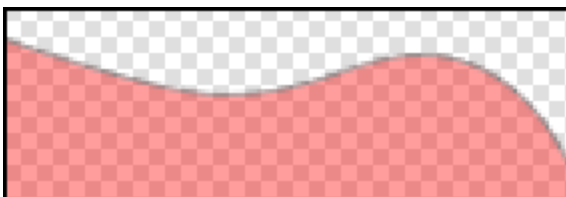
The Set Color code block sets the sprite's current costume to the red, green, and blue values supplied as an input to the procedure.



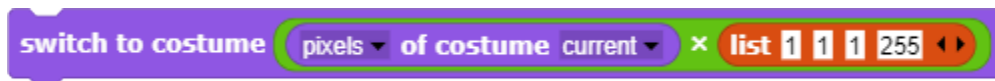
This makes it possible to adjust the colors of any costume to the specified RGB values.

Adjusting Transparency

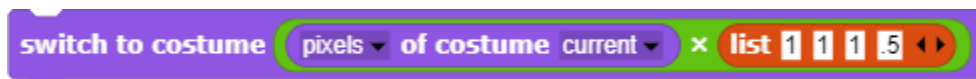
In addition to specification of the red, green, and blue values for each pixel, a fourth parameter controls the transparency of the pixel. When the transparency parameter is set to 0, the pixel is completely transparent (i.e., invisible). When the transparency parameter is set to 255, the pixel is completely opaque. Values between 1 and 255 result in varying degrees of transparency. In the magnified image below, the checkboard squares that are white or gray are pixels that have a transparency value of 0. Pixels that are darker have a higher level of opacity.



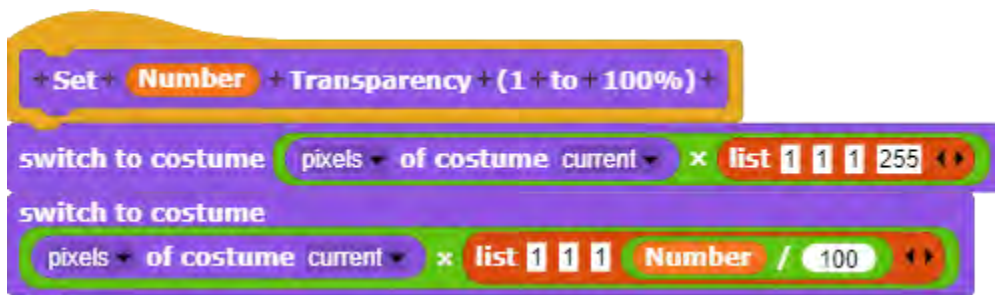
To adjust the transparency of the image, an operation is required that leaves all of the pixels that have a value of zero unchanged, while adjusting the values of the pixels that have a non-zero value. This can be accomplished through multiplication. In the example below, the first three parameters (red, green, and blue) are multiplied by 1. This leaves their values unchanged. The fourth parameter, transparency, is multiplied by 255. This operation makes all of the pixels with a non-zero value complete opaque, while leaving the pixels which have a transparency value of zero untouched (since multiplying zero by 255 results in zero (i.e., $255 \times 0 = 0$)).



Once all of the pixels with non-zero transparency values are set to maximum opacity, the desired level of transparency can be specified. In the example below, the transparency levels of the pixels have been set to 50%. Since multiplying 255 times 0.5 results in a rounded result of 128, the resulting level of transparency is halfway between completely opaque and completely transparent.



The Set Transparency code block employs this method to enable a sprite's costume to be set to a specified level of transparency.



In the example below, the transparency level of the image has been set to 50 percent.



The advantage of creating a custom Set Transparency code block is that it is no longer necessary to remember the details of the internal pixel structure and parameters while focusing on other aspect of the program.

Appendix B

New APCS Principles

Topic 5.1 Animating Objects

CS Principle CSN-2.A Parallel Computing

Parallel computing is a method of computing where the program is broken into multiple smaller computing operations, some of which are then performed simultaneously. This is different from the programs used in previous modules, which relied on sequential computing. In sequential computing, operations are performed in order and one at a time.

Application

In Snap!, the Green Flag block can be used to start multiple programs or statements in parallel with one another. With parallel computing, multiple sprites can run multiple code segments at the same time, without having to wait for each other to finish. This would be impossible using only sequential computing.

Topic 5.2 Exploring RGB Color

CS Principle CRD-2.J Identify Expected Outputs

In the development process, testing uses defined inputs to ensure that an algorithm or program is producing the expected outcome. Programmers use the results from testing to revise their work. Defined inputs used to test a program should demonstrate the different expected outcomes that are at the extremes of input data.

Application

When adjusting RGB color settings, each variable can be set to its minimum value (0). If the variables are changed one at a time to their maximum values (255), students should be able to accurately predict resulting color of the sprite. Once students have verified that the variables produce the desired results independently, they can try adjusting two or more sliders to try blending colors. This will illustrate the importance of verifying that a program is working as expected with regard to its inputs and outputs.

Topic 5.3 Adjusting Color

CS Principle CRD-2.F User Interface

When a program is being developed, it's important to understand not only the program, but also the concerns and interest of the people who will use it. Program requirements describe how a program functions and may include a description of user interactions that the program must provide. Some ways to investigate these needs include user testing, interviews, and direct observation.

Application

In the previous section, it was demonstrated how colors can be changed by adjusting the numerical values of the RGB settings. In that section, these adjustments were made manually. As the procedure to set the color gets integrated into the larger program, it may be easier for users if they didn't have to directly edit the code to change colors. In this module, slider controls are created to allow users to more easily adjust sprite colors.

Topic 5.7 Pulsing Sounds

CS Principle AAP-2.E Relational Operators

Relational operators include the following symbols: =, ≠, <, >, ≤, ≥. These are used to test the relationship between two variables, expressions, or values. A comparison using a relational operator

evaluates to a Boolean value. For example, $a = b$ evaluates to true if “a” and “b” are equal; otherwise, it evaluates to false.

Application

In this topic, the greater than symbol is used to determine whether the length of the input list has been exceeded. If the length of list has been exceeded, the procedure begins again with the first item in the list.

CS Principle AAP-2.H Conditional Statements

Conditional statements, or “if-statements”, affect the flow of a program by executing different statements when different conditions are met. If the condition of the if-statement is determined to be true, the code included in the if statement will be executed. If the condition is determined to be false, the program or procedure will ignore the if statement and continue to the next statement in the code sequence.

Application

In this section, a procedure repeats a sequence of sounds using a For loop. Sounds are numbered sequentially, and the code in the loop plays the sounds in order. To make the sequence repeat, a conditional statement is added, telling the procedure that if the number of the sounds set to play is greater than the number of sounds that exist, it should start back over with the first sound.

CS Principle DAT-1 Data

The way a computer represents data internally is different from the way the data are interpreted and displayed for the user. Programs are used to translate data into a representation more easily understood by people

A. Explain how data can be represented using bits.

1. Data values can be stored in variables, lists of items, or standalone constants and can be passed as input to (or output from) procedures.
2. Computing devices represent data digitally, meaning that the lowest-level components of any value are bits.
3. Bit is shorthand for binary digit and is either 0 or 1.
4. Bits are grouped to represent abstractions. These abstractions include, but are not limited to, numbers, characters, and color.
5. Analog data have values that change smoothly, rather than in discrete intervals, over time. Some examples of analog data include pitch and volume of music, colors of a painting, or position of a sprinter during a race.
6. The use of digital data to approximate real-world analog data is an example Abstraction.
7. Analog data can be closely approximated digitally using a sampling technique, which means measuring values of the analog signal at regular intervals called samples. The samples are measured to figure out the exact bits required to store each sample.