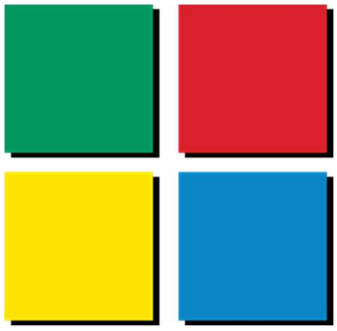
**Musical Games**

The musical game *Simon*, introduced in 1978, initiated an era of electronic games (Smithsonian Magazine). A series of tones light up panels on the game. The player must repeat the sequence by touching each panel in the correct order. The musical sequence becomes progressively longer and more complex. The game was an immediate hit and is still sold today.


Electronic games were made possible by small, affordable computer chips. The computer chip in the *Simon* game synthesized musical tones just as *TuneScope* does.

The computing power underlying applications like *TuneScope* makes it possible for anyone with the interest to create musical games. The TuneScope version of the musical sequencing game is recreated through creation of a green, red, yellow, and blue sprite.



Each sprite has two costumes: a solid-colored costume, and a second costume with a radial gradient that begins with a solid color at the perimeter and fades to a yellow shade in the center. The sprite switches to the second costume when a tone is played.

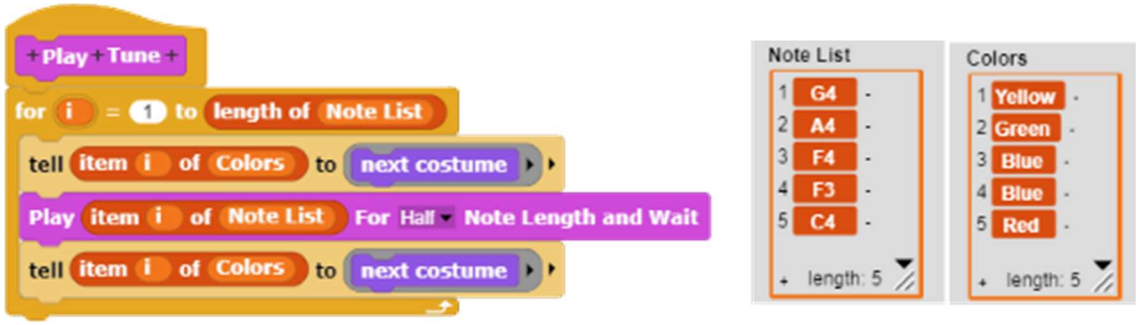
The **Switch to Costume** code block is used to switch to the lighted costume when the right button of the mouse is pressed while the mouse is over the sprite. When the mouse button is released, the sprite plays a note and then switches back to the solid costume.


In the movie *Close Encounters of the Third Kind* scientists use a music synthesizer to communicate with the aliens' space ship:

- Start with the note - "G"
- Up a full note - "A"
- Down a major third – "F"
- Now drop an octave – "F" (an octave lower)
- Up a perfect fifth – "C"

Because the space ship in the movie resembled the musical game, the movie contributed to the popularity of the game. Therefore, it seems appropriate to use this series of notes in this illustration. The now-familiar **For** loop is used to play a list of notes contained in the variable *Note List*.


Each note in the *Note List* has a parallel color associated with it in a parallel list of *Colors*. The **Play Tune** procedure is extended to make use of the colors. The sprite of the corresponding color is told to switch to the next costume before the note is played. For example, before the first note in the *Note List* (G4) is played, the *Yellow* sprite is told to switch to the next (highlighted) costume. After the note is played, the *Yellow* sprite is told to switch back to the solid costume.




As a result, the sprite associated with each note is highlighted when the note is played.

A **Play Game** procedure is used to play the tune, lighting up the associated squares as each note is played. The player must click each square in the same order as the notes in the original tune.

Two additional blocks are added to each sprite's script (i.e., the scripts associated with the green, red, yellow and blue squares). The **Set Color** code block sets a variable named *Color* to the color of the block. The **Set Check Color?** block then sets a second variable named *Check Color* to a value of *True*. (The **True / False** block is found in the *Operators* section of the *Code Block Palette*.)


The **Play Game** procedure first plays the tune, lighting up the corresponding squares, and then sets the value of the *Check Color?* variable to *False*. This will cause the procedure to pause at this point, waiting until a square is clicked.


Clicking one of the squares then changes the value of the *Check Color?* variable to *True*, enabling the **Play Game** procedure to advance to the next group of code blocks.

This group of blocks consists of a *For* loop that waits until the value of the *Check Color?* variable is changed to *True*. When a square is clicked, an *If* statement is used to check whether the color of the square clicked is the same color as the corresponding item in the list of colors. If the colors do not match, an *End Game* sound is played and the **Stop All** code block is used to end the game.

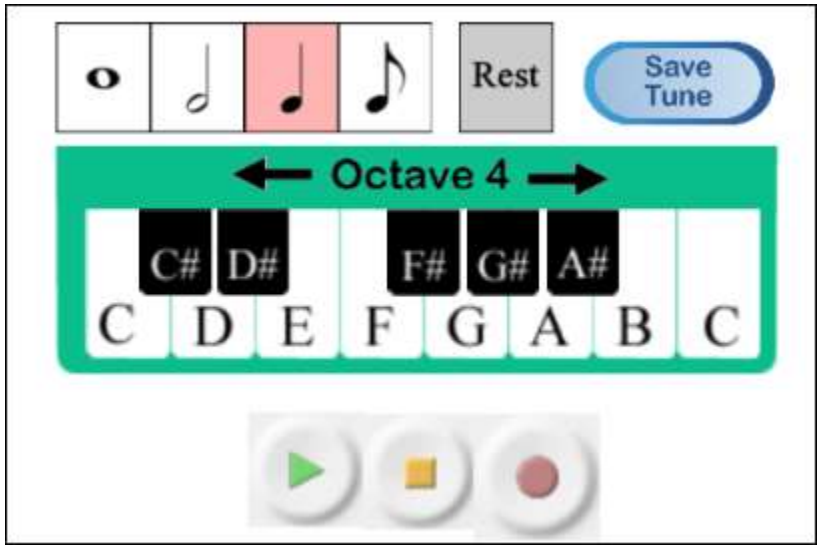

If the colors do match, the *Check Color?* variable is reset to *False* again. The **For** loop then waits until the next square is clicked, setting the *Check Color?* variable to *True*.If all the squares are selected in the correct order, a "You win!" sound (a trumpet fanfare) is played.

The complete **Play Game** procedure looks like this. The **Play Game** procedure starts the game and then waits until a square is clicked. If the color of the square clicked is correct, the procedure waits until the next square is clicked. The game ends either when an incorrect square is clicked, or when the player successfully completes the entire sequence in the correct order.


This basic musical game procedure could be adapted and refined. For example, the game could be modified so that it initially plays a single note, and then adds an additional note each time the player is successful.

One limitation of the game is that it can only be used with four notes. (The notes "F4" and "F3" are both assigned to the blue square in this illustration.) The number of notes could be extended through variants of the game such as creation of a musical version of the "Whack a Mole" game. In this variant, moles could pop up as each note was played. (In another variation, memorizing a tune might allow a player to anticipate where the next mole is scheduled to pop up.)

A similar technique could be used to create a player piano such as the one shown below.




Once a tune was recorded, the player could be challenged to play the same sequence of notes using the keys on the piano keyboard.