Program Design

Link to Code Showcase: https://youtu.be/nbEgITsEtGY
Link to Animation Demonstration: https://youtu.be/p5zotoWNo_E

Purpose

The purpose of this program is to demonstrate how Splashkit can be used to make short animations. Inspired by the famous gun barrel sequence that can be seen at the beginning of nearly every James Bond film, this program recreates it using C++ combined with Splashkit.

Design

- ### Sequencing System

  The Gunbarrel program uses a sequencing system to play the gun barrel animation. What this sequencing system does is the programmer splits the animation into shorter sequences and then writes the procedure for that sequence of animation (i.e.: tell a circle to move across the screen and create a trailing circle if at a certain distance across). The programmer then decides on a conditional statement that decides when that sequence should be complete, after which the next animation sequence can begin and the process repeats until all sequences of the animation have been completed.

  Key procedures: Sequence1, sequence2, sequence3, sequence4, sequence5, sequences (found in rendering.h/cpp)

  These procedures represent part of the full gun barrel animation. These are executed in chronological order and represent what is meant to happen in one frame of sequence. The sequence procedure is executed many times, changing what is on the screen every time, until its complete condition is true and the next sequence procedure can begin executing.

  Key datatype: sequence_progress (found in program.cpp)
  Key variable: sequence_tracker (found in datatypes.h)

  This is an array that stores elements of the type 'sequence_progress'. Sequence_progress is a struct that consists of a pair of Boolean values. The first Boolean tracks whether a sequence of animation is complete or not while the second Boolean tracks whether the graphics and shapes for a sequence has been loaded into the drawing arrays. The sequence_tracker array stores as many of these sequence_progress structs as there are sequences that make up the animation (in Gunbarrel, there are 5 sequences, so therefore, there will be 5 sequence_progress elements stored in the array). The sequence of animation that each element keeps track of corresponds to its index location in the array, so the sequence_progress element at index 2 will keep track of whether sequence 3 is complete and whether the graphics for it have been loaded. These sequence procedures are then wrapped up into a procedure called sequences, which, given a switch statement and a number, executes the corresponding sequence in question.

  In program.cpp, a for loop can be found that loops through the sequence_tracker array until it finds an element whose complete Boolean is set to false. When it finds this, it will then call

the sequences procedure and tell it to call its corresponding sequence procedure using the index value of the sequence_tracker element. When all the elements in the array have their complete Booleans set to true, this implies that the animation is finished.

- ## Rendering System

The rendering system for Gunbarrel is a very conceptually simplistic system. The programmer declares some arrays for each type of graphic or shape, populates them with data and then tells the computer to render everything from those arrays onto the screen.

Key datatypes: circle_properties, rectangle_properties, bitmap_properties

The circle_properties, rectangle_properties and bitmap_properties structures hold the information that is required throughout the animation for each individual graphic on the screen (i.e.: location, colour or name of image). These are loaded into the arrays of their corresponding data type using their corresponding populate procedures.

Key variables: circles, rectangles, bitmaps (found in program.cpp)
Key procedures: populate_circles, populate_rectangles, populate_bitmaps, populate (found in rendering.h/cpp)

The populate graphics procedures take the information needed to form a properties structure element as the parameters. It then takes all those parameters and creates a properties structure consisting of them and stores it in the array for rendering. These populate graphics procedures are wrapped up into the populate procedure which works by deciding which graphics and shapes to load into the arrays. This is decided based on which sequence the program is currently executing, so it requires access to the sequence_tracker array for this reason. Since the populate procedure is called as part of the sequences procedure, it takes the sequence number that has been passed in from the for loop in program.cpp to know which sequence it is dealing with. After that, it checks the graphics_loaded Boolean value of the corresponding sequence_progress element in the sequence_tracker array. If it is false, then the graphics have not been loaded yet and will proceed to load them into the arrays, otherwise, it does nothing. When the graphics have been loaded, the graphics_loaded value changes to true to prevent the program from loading the graphics into the arrays repeatedly.

Key procedures: render_circles, render_rectangles, render_bitmaps, render (found in rendering.h/cpp)

The render graphics procedures simply take everything from their corresponding arrays and draws them onto the screen. The render_bitmaps procedure works slightly differently, however. Render_bitmaps renders bitmap images onto the screen starting from the end of the bitmaps array rather than at index 0. This is so that the programmer can make logical sense of the order in which images are rendered onto the screen. When drawing things onto the screen using code, the order in which things are drawn will determine which things are drawn over the top of one another. With this in mind and in order to have better control over this, the images that are at the end of the array are considered as background images while the ones more towards the index 0 are considered as foreground images. Ultimately, this is so that foreground images can be rendered over the top of background images and so that the order can easily be modified when new images enter the array. The render graphics procedures are also wrapped up in the render procedure, which simply executes all three of the render graphics sequences.