SIT102 Introduction to Programming

Pass Task 3.2: Shape Shifter

Overview

Now that we have control flow and variables, we can create a program which will react to user input! In this task you will create a program that will allow the user to move a circle around on the screen.

Submission Details

Use the instructions on the following pages to create your own Shape Shifter program.

Submit the following files to OnTrack.

- Your program code
- A screen shot of your program running

The focus of this task is on control flow and the use of loop and branching statements to make more interactive programs.

Background

So far all of the SplashKit programs we have built contained a sequence of actions (such as shape drawing) controlled by delays. However, this situation does not allow the user to interact with the program (the program is ended by a hard coded delay). What if the user wanted to look at the drawing for longer than specified in our delay, or for less time than in our delay? What we need is for the window to remain open until the user requests it to be closed.

Another common feature that a user expects in programs like games is to be able to move things around. For example, if the user presses the move right arrow on the keyboard for this task then the circle moves to the right. In this task you will create a program that uses control flow mechanisms (such as do/while and if/else) to allow the program to respond to user input (such as closing a window, keyboard actions, and mouse actions).

To do this we will be creating an event loop which calls procedures to recognise and respond to user input (such as closing windows when requested and moving circles using the keyboard). Table 1 shows some of SplashKit's functions and procedures for working with input.

Function/Procedure	Action / Check
`process_events();`	Listens for user input. No input can be received unless this is called.
`double mouse_x();`	X location of the mouse. A `mouse_y()` function also exists.
`bool mouse_clicked(button);`	Was the mouse button clicked?
`bool quit_requested();`	Has the user asked to close the program?
`bool window_close_requested();`	Has the user asked to close the Window?
`bool key_down(key);`	Is the key currently held down?
`bool key_typed(key);`	Was the key typed? (Pressed then released)

There are also some new values you can use related to keys (the `key_code` type from SplashKit) and mouse buttons (the `mouse_button` type from SplashKit).

- `key_code` values are in the format `..._KEY` or `NUM_..._KEY` for the numeric keys. For example: `A_KEY`, `LEFT_KEY`, `UP_KEY`, `SPACE_KEY`, `NUM_1_KEY`, etc.
- `mouse_button` values are in the format `..._BUTTON`. For example `LEFT_BUTTON` and `RIGHT_BUTTON`.

The following code demonstrates how you could check if the escape key was typed, and how you could check if a mouse button was clicked.

```
if ( key_typed(ESCAPE_KEY) ) ...
if ( mouse_clicked(LEFT_BUTTON) ) ...
```

Instructions

Watch the 3.2P Circle Mover video and follow the steps there. The following steps outline the details shown on the video, so either follow the steps in the video or use the following as a guide.

Skip to the end for the additional instructions in the **Your Task** section if you have followed the steps in the video.

1. Start by creating a new folder to store your project's code, and initialising it with a SplashKit c++ project. The following shows the instructions for doing this in the terminal.

   ```
   cd /c/Users/andrew/Documents/Code
   mkdir CircleMoving
   cd CircleMoving
   skm new c++
   ```

2. Open **Visual Studio Code** and open the *folder* you created.

   Remember to make sure to open the **folder** (directory) not just the file. This will let VS Code find all of the files related to your project, including the SplashKit files.

3. Get started by adding the code to open a window, clear its contents, refresh, and delay.

   ### Pseudocode for main

   Returns: int

   Steps:

   ```
   ◦ Open a Window with t itle "Shape Shifter" that is 800 x 600
   ◦ Clear the Screen to COLOR_WHITE
   ◦ Refresh the Screen limiting it to 60 FPS
   ◦ Delay for 5000 milliseconds
   ◦ Return 0
   ```

4. To compile this program you need to tell the compiler about the C/C++ code file:

   ```
   skm clang++ program.cpp -o CircleMoving
   ./CircleMoving
   ```

   Notice the window closes after 5 seconds, as the program's instructions end.

5. To give the user more control over this we can add an event loop, a loop that will repeat our instructions over and over checking what the user wants done each loop.

    - The event loop will be located in main, and will loop until the user asks to quit the program (by closing the window).
    - The `process_events` procedure needs to be called once each loop to update SplashKit with the actions that have occurred since the last time the loop was executed.

    Update your code to implement the following pseudocode in your main.

    Pseudocode for main

    Returns: int

    Steps:

    ```
    Open a Window with title "Shape Shifter" that is 800 x 600
    Do
        Process Events
        Clear the Screen to COLOR_WHITE
        Refresh the Screen limiting it to 60 FPS
    While Not Quit Requested
    Return 0
    ```

6. Switch back to the Terminal and compile and run the program. It should now remain open until you close the window.

    The event loop repeats code over and over as the program runs. This now means that you can create interactive programs. Inside this loop you can listen for user events and then update the program's data. Redrawing the screen will then show the user what has changed.

7. Alter Main to draw a circle on the screen, using variables for the circle's x and y location.

Pseudocode for main

Returns: int

Steps:

```
◦ Declare local variables for x, and y to store the circle's location (use the double data type)
◦ Open a Window with title "Shape Shifter" that is 800 x 600
◦ Assign x the value 400
◦ Assign y the value 300
◦ Do
    ▪ Process Events
    ▪ Clear the Screen to COLOR_WHITE
    ▪ Fill a Circle using COLOR_GREEN, at location x, y with a radius of 100
    ▪ Refresh the Screen limiting it to 60 FPS
◦ While Not Quit Requested
◦ Return 0
```

8. Switch to the Terminal, compile and run the program. You should be able to see a green circle in the centre of the screen.
    - The x and y variables in Main store all of the data for this game: the location of the circle. As these are the only variables in the "game" (which is being run by the steps in the Main procedure), these are the only things that can change.

The next step will involve using if statements to selectively run sections of your code. This can be used to ensure the computer only runs certain code when a condition is met. For example, we can only move the circle to the left when the left arrow key is held down.

9. Alter Main to update the x variable when the user is holding down either the left or right arrow. This will...
    - Make x smaller if the user is holding down the left arrow key, which will move the circle left
    - Make x larger if the user is holding down the right arrow key, which will move the circle right

Pseudocode for main

Returns: int

Steps:

```
◦ Declare local variables for `x`, and `y` to store the circle's location (use the `double` data type)
◦ Open a Window with title "Shape Shifter" that is `800` x `600`
◦ Assign `x` the value `400`
◦ Assign `y` the value `300`
◦ Do
    ▪ Process Events
    ▪ if the `LEFT_KEY` Key is Down then
        ▪ Assign `x`, the value `x - 1`
    ▪ if the `RIGHT_KEY` Key is Down then
        ▪ Assign `x`, the value `x + 1`
    ▪ Clear the Screen to `COLOR_WHITE`
    ▪ Fill a Circle using `COLOR_GREEN`, at location `x`, `y` with a radius of `100`
    ▪ Refresh the Screen limiting it to `60` FPS
◦ While Not Quit Requested
◦ Return 0
```

10. Switch back to the Terminal and compile and run the program. You should be able to move the circle using the left and right arrow keys.

11. Add code to use `UpKey` and `DownKey` to move the circle along the Y axis (up and down the screen).

12. Switch back to the Terminal and compile and run the program. You should be able to move the circle left, right, up, and down using the arrow keys.

13. If you keep your finger on the one arrow key long enough, the circle will disappear off the edge of the screen.

14. Before addressing this problem, lets fix the use of the literal value `100` as the radius. Instead, add a constant named `RADIUS` and set it to `100`.

    TIP:

    Avoid having "magic numbers" in your code. Numbers like 100 do not have as much meaning as a constant like `RADIUS`. Later when you read or change the code these constants will make it easier to understand and change.

15. Alter main to use the `RADIUS` constant when it draws the circle.

16. Now, adjust the program to ensure that the circle remains on the screen.

    TIP:

    To find the width of the current screen you can call the SplashKit function `screen_width()`, and you can use `screen_height()` to get the height of the screen.

    HINT:

    When moving right you should only change the x value if the key is held down and x is less than `screen_width() - the circle's radius`.

    Eg:

    ```
    if ( key_down(RIGHT_KEY) &&
         x < screen_width() - RADIUS )
    {
        ...
    }
    ```

17. Switch to the Terminal. Compile and run the program, and test that you cannot move the circle off the screen to the left or right.

18. Now, change the radius of the circle to be `150`. Compile and run the program again, and if you have coded it correctly the circle should still be able to go right up to the edge of the screen.

Your Task

Now you have the code from the video working... here are some changes for you to make:

1. Add the ability to move along the `y` axis using the up and down arrow keys.
2. Ensure that you cannot move above the top or below the bottom of the window.
3. Make the radius a variable and allow the user to change this with the + and - keys. (In the code

you may need to use the equals key instead of plus, as + is a combination of shift and equals)

4. Include constants for a minimum and maximum radius (set minimum radius to 10 and maximum radius to 200) and make sure the user cannot adjust the radius outside of this range.
5. Add the ability to reset the circle to the center of the screen.

Grab a screenshot of your program running, and upload it along with your code to OnTrack.

Task Discussion

Discuss the following with your tutor to demonstrate your understanding of the concepts covered.

- Focus on explaining how control flow enables your program to respond to events. How do sequence, selection, and repetition come together here to make this possible?
- This is also an opportunity to explain the how the control flow mechanics work.