Faculty of Science, Engineering and Technology

Introduction to Programming

Pass Task 2.2: My Picture Drawing

Overview

Create a program that calls procedures to draw a picture to a window.

Purpose:	Create a graphical program to explore the use of procedures, and to understand that instructions are run in sequence.
Task:	Create your own Picture Drawing program that draws something other than a house which we use as the example. Submit to Doubtfire when complete.
Time:	This task should be started in your third lab class and submitted for feedback before the start of week 5.
Resources:	▪ Chapter 1 of the Programming Arcana ▪ Swinburne CodeCasts (YouTube Channel, iTunesU) • Understanding Syntax Rules • Programs and Sequence ▪ Syntax Videos • Introduction, Getting Started, Calling Procedures, and Creating Your Own Procedures

Submission Details

You must submit the following files to Doubtfire:

- Picture Drawing source code (GameMain.pas)
- Screenshot of the Window showing your picture.

Make sure that your task has the following in your submission:

- Your picture has at least 4 shapes, and is something other than a house.
- Code layout - match the example for indentation and use of case.
- The code must compile and the screenshot show it working on your machine.
- The program must show a picture of some kind, bonus points for some creativity.

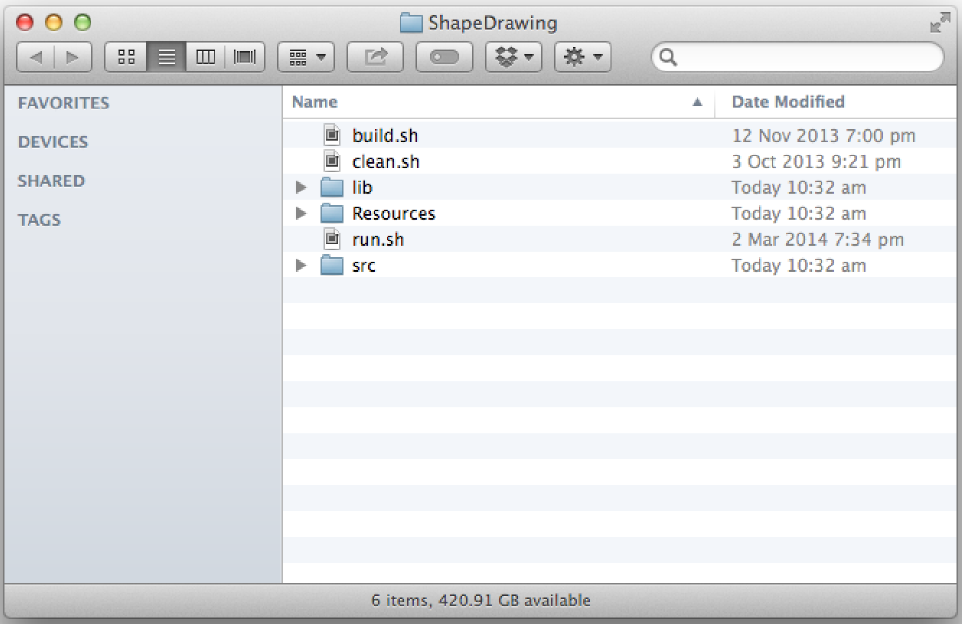
Instructions

For this task you will create a program that draws a scene using primitive shapes (triangles, rectangles, and circles).

The goal of this exercise is to learn a little about how to create a program using the *SwinGame* Software Development Kit (SDK). SwinGame is a development environment that makes it easy to create programs that use graphics, sounds, animations, networking, and other aspects relevant to creating small interactive games.

When building a program using *SwinGame* you need to start with a *SwinGame Project Template.* This template contains the necessary development environment and code that you can make use of.

1. Download the **Pass Task 2.2 code** from the Resources for this task. This contains everything you need to create your own programs that use the SwinGame SDK.
2. Extract the zip file to your code directory (e.g. Documents/Code)
3. Open the PictureDrawing folder and you should see the following files:
   - The **clean.sh**, **build.sh** and **run.sh** scripts. These contain bash commands you will use to clean up the project, compile (or build) the project, and run your program.
   - The **src** folder contains your program's source code
   - The **lib** folder contains the *SwinGame* code
   - The **Resources** folder contains images, sounds, and other resources you want to use.

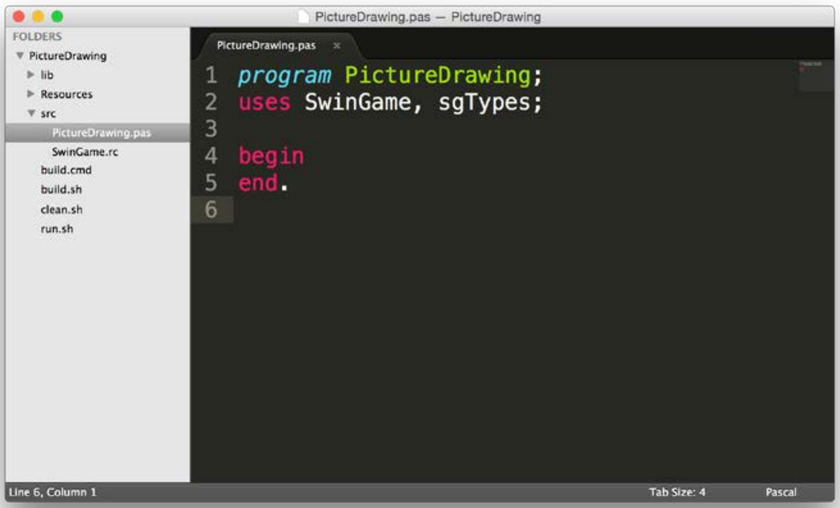


Name	Date Modified
build.sh	12 Nov 2013 7:00 pm
clean.sh	3 Oct 2013 9:21 pm
lib	Today 10:32 am
Resources	Today 10:32 am
run.sh	2 Mar 2014 7:34 pm
src	Today 10:32 am

6 items, 420.91 GB available

4. Open the **PictureDrawing.pas** file in the **src** folder using Sublime Text. It is a little empty at the moment, but the uses SwinGame code gives you access to the SwinGame library.

Tip: When you first open a .pas (or .sh) file from Windows Explorer or Finder, your operating system might not know what program to open it with. It is handy to set the default program for .pas files to your favourite text editor. See the Top Tips Guide for how to do this.



```
program PictureDrawing;
uses SwinGame, sgTypes;

begin
end.
```

5. Compile and run the current code in PictureDrawing.pas by doing the following:
   - Open a **Terminal** window (or use one you already have open)
   - Change into the *PictureDrawing* directory using the **cd** command
   - Compile the template program using **./build.sh**
   - Run your program using **./run.sh** … it doesn't do much but it should start and stop without error.

Note: The **./build.sh** is a Bash script, it is a saved list of commands you could type at the terminal yourself - this just saves you time. This Bash script was written especially for compiling *SwinGame*. It calls the compiler for you, as well as copying across the files you need for the program to run.

Tip: Although the code is stored in the **src** directory, you need to compile and run *SwinGame* programs from the directory containing the **build.sh** and **run.sh** commands.

Before adding to the program we need to review how drawing works in computers, and how you can draw using SwinGame.

Opening a Graphics Window

In *SwinGame* you can open a **Window** to draw into. To open the window you call the **Open Graphics Window** procedure, and pass it the window's title, its width and its height. For example

```
OpenGraphicsWindow('House Drawing', 800, 600);
```

will open a graphics window 800 px wide and 600 px high with the title "House Drawing" (see Figure 1. Please note that the house and hill are drawn by additional code).

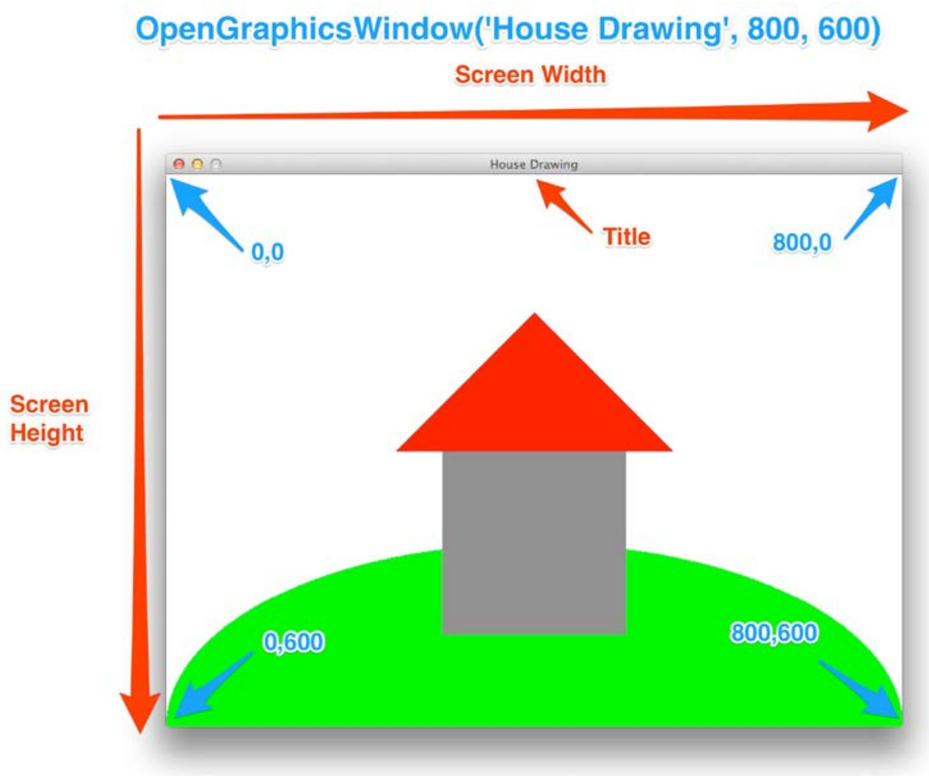


Figure 1: Example of SwinGame graphics window and house drawing

Pixels and Shape Drawing

The images you see on your computer's screen are made up of *dots* called **pixels**: picture elements. The screen has many pixels arranged into a grid (rows and columns), with each pixel having its own unique location (a combination of an **x** and **y** value), and **color** (Note: In programming we will be using the American spelling of colour).

Figure 2 shows an example of two rectangles (one filled, one outlined). The top left corner of the screen is at row (x) 0 and column (y) 0, and these numbers increase as you go to the right and down the screen.

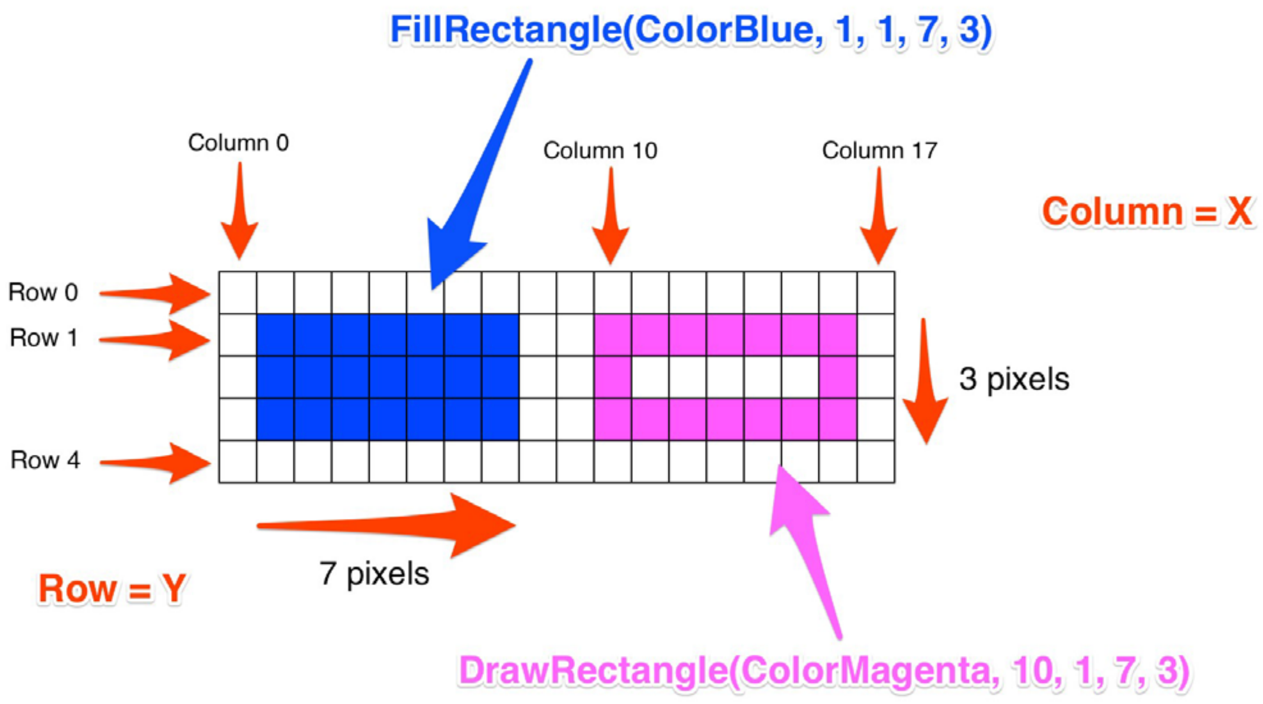

FillRectangle(ColorBlue, 1, 1, 7, 3)
Column 0
Column 10
Column 17
Column = X
Row 0
Row 1
Row 4
3 pixels
7 pixels
Row = Y
DrawRectangle(ColorMagenta, 10, 1, 7, 3)

Figure 2: Pixels are organised into columns (x) and rows (y).

Positions on the screen are determined using two values, one for x and the other for y. The x value determines the number of pixels from the left side of the screen. The y value determines the number of pixels from the top of the screen.

For example: the magenta rectangle is drawn at 10, 1. This means its x value is 10 and its y is 1. This rectangle is drawn 10 pixels from the left of the screen, and its 1 pixel from the top.

To draw a shape on the screen with *SwinGame* you need start by telling the computer what color to draw it and where to draw it (i.e., the x y coordinates). Different shapes will require additional information such as *width* and *height* for rectangles and *radius* for circles.

All of the shape drawing operations in *SwinGame* take a number of parameter values:

- The **color** to draw the shape. The built in colors you can use include ColorWhite, ColorGreen, ColorBlue, ColorBlack, ColorRed, ColorYellow, ColorPink, ColorTurquoise, ColorGrey, ColorMagenta, ColorLightGrey and many others.
- An **x** value, representing the x position of the shape (column). This is a number of pixels from the left edge of the screen. Larger values are further to the right.
- A **y** value, representing the y position of the shape (row). This is a number of pixels from the top edge of the screen. Larger values are further down the screen.
- Values for the size of the shape, these will differ depending on the kind of shape being drawn (e.g., rectangle has a *width* and *height,* as does ellipse).

Table 1 shows the parameters for the different shapes you can draw with SwinGame, an example procedure call is also shown.

Procedure	*Example*
ClearScreen(color)	ClearScreen(ColorWhite);
DrawCircle(color, x, y, radius)	DrawCircle(ColorRed, 50, 100, 25);
FillCircle(color, x, y, radius)	FillCircle(ColorRed, 50, 100, 20);
DrawRectangle(color, x, y, width, height)	DrawRectangle(ColorGreen, 100, 150, 30, 60);
FillRectangle(color, x, y, width, height)	FillRectangle(ColorGreen, 100, 160, 10, 40);
DrawTriangle(color, x1, y1, x2, y2, x3, y3)	DrawTriangle(ColorBlue, 150, 100, 150, 200, 175, 200);
FillTriangle(color, x1, y1, x2, y2, x3, y3)	FillTriangle(ColorBlue, 155, 135, 155, 195, 170, 195);
DrawEllipse(color, x, y, width, height)	DrawEllipse(ColorBlack, 200, 100, 50, 160);
FillEllipse(color, x, y, width, height)	FillEllipse(ColorBlack, 210, 110, 30, 140);
DrawLine(color, x1, y1, x2, y2)	DrawLine(ColorMagenta, 0, 300, 800, 300);

Table 1: SwinGame shape drawing procedures and their parameters

Displaying Shapes using Double Buffering

In a House Drawing program like the one used to create Figure 2, the computer executes the code to draw the individual shapes one at a time in the order they appear in the code. However, we don't want each element to appear individually, we just want the whole house to appear at once. *SwinGame* uses a technique called **Double Buffering** to enable this. When double buffering, the computer first draws the shapes, then waits for a command to display the shapes to the user. With SwinGame, the shapes are all shown together when the program calls the **Refresh Screen** procedure. See Figure 3 for an illustration of how this procedure should be used.

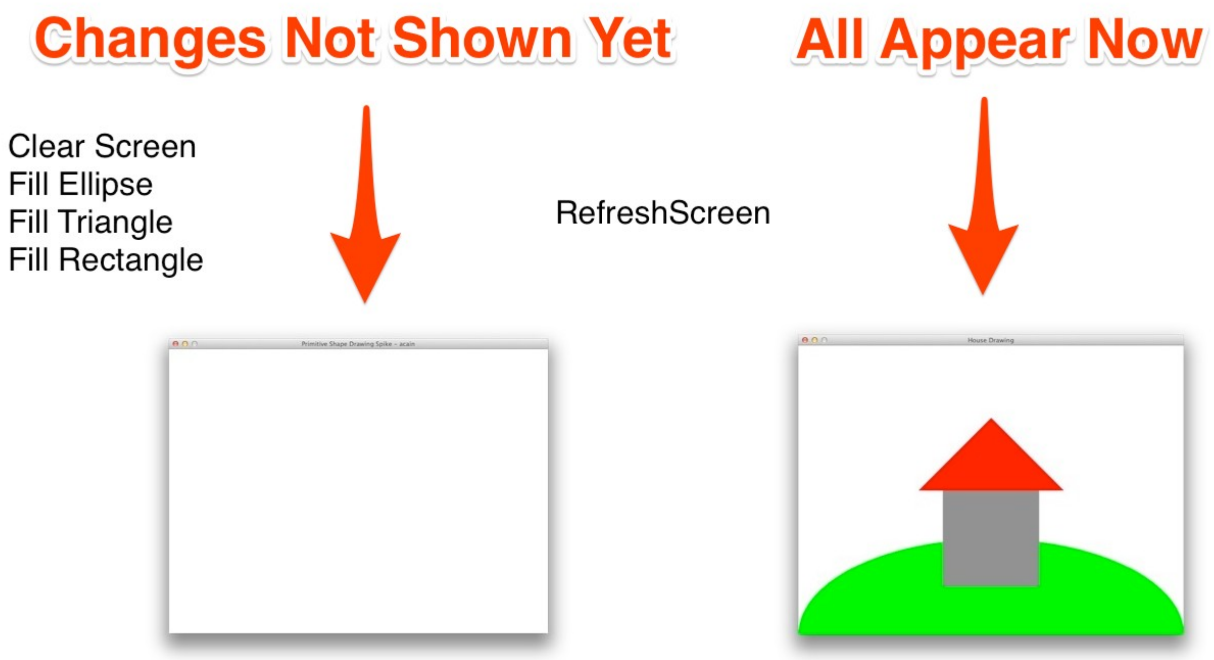


Figure 3: Using RefreshScreen in SwinGame

The house drawing program shown in Figure 2 and 3 is complete once the shapes are drawn. To keep your program running a little longer you can add a call to the **Delay** procedure. This will cause the computer to wait a specified number of milliseconds (1000th of a second) before it executes the next command. For example, use **Delay(5000)** to have the program wait 5 seconds before it goes to the next instruction.

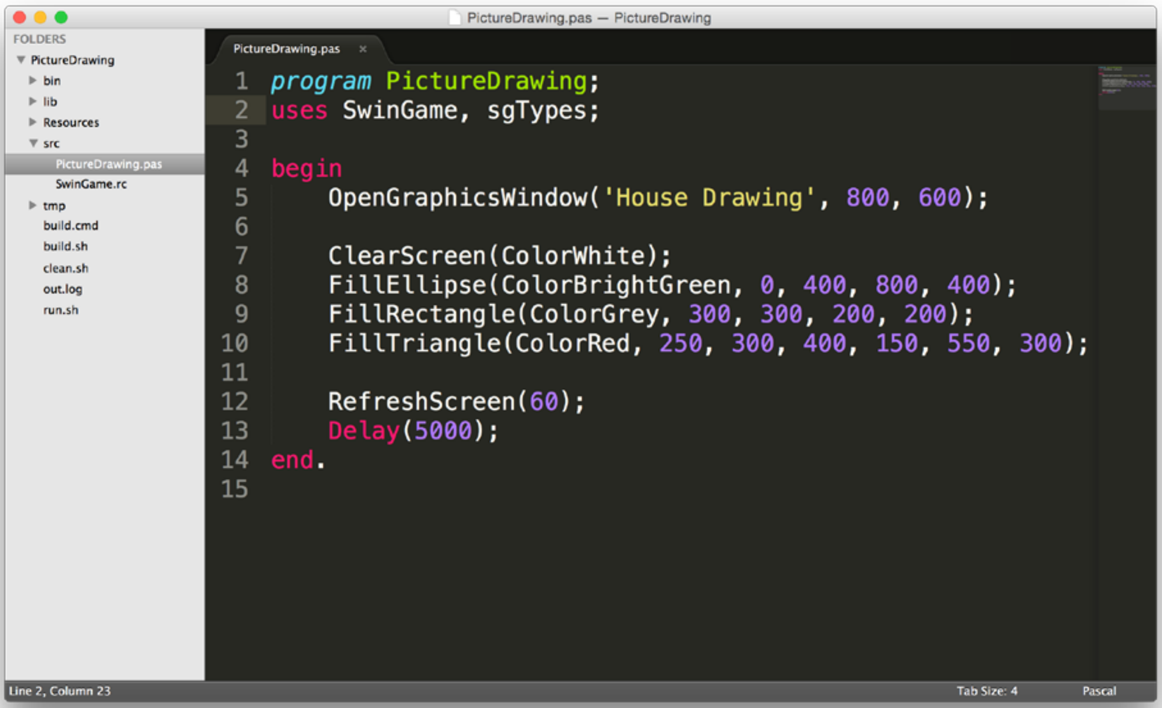
Tip: Using a Delay will help you take a screenshot before your drawing disappears!!

In summary, to display a drawing like that shown in Figure 2 using *SwinGame* requires a program that executes the following steps.

1. Clear the screen of any previous drawing (using **ClearScreen()**)
2. Draw the shapes (e.g., Fill Ellipse, Fill Triangle, Fill Rectangle)
3. Display the shapes (using **RefreshScreen()**)
4. [Optional] Delay executing the next command (e.g., **Delay**(5000))

Tip: Black is the default `ClearScreen()` colour. Add a colour parameter to change the screen colour (e.g., `ClearScreen(ColorWhite)` will draw a white screen).

Here is some example code that draws a house. This code demonstrates the convention you need to use for writing Pascal programs - this covers the general look and feel of the code. Notice the keywords like program, uses, begin, and end are all in **lowercase**. The program, unit (SwinGame) and procedure identifiers (eg: OpenGraphicsWindow) are all in **PascalCase** where the first letter of each word is uppercase and the others are lowercase. Also notice the indentation. The instructions within the program are all tabbed in four spaces from the left, see how this makes it easier to see that they belong inside the program (inside its begin … end code).



```pascal
program PictureDrawing;
uses SwinGame, sgTypes;

begin
    OpenGraphicsWindow('House Drawing', 800, 600);

    ClearScreen(ColorWhite);
    FillEllipse(ColorBrightGreen, 0, 400, 800, 400);
    FillRectangle(ColorGrey, 300, 300, 200, 200);
    FillTriangle(ColorRed, 250, 300, 400, 150, 550, 300);

    RefreshScreen(60);
    Delay(5000);
end.
```

Create your own program that draws a picture using the procedures from Table 1. You must draw something other than a house, for example you could draw a car with some rectangles and circles.

Once you are happy with your Picture Drawing program you can prepare it for your portfolio. This work can be placed in your portfolio as evidence of what you have learnt.

1. Use Skitch (or your preferred screenshot program) to take a screenshot of your program in action.
2. Login to Doubtfire and submit your code and screenshot to Pass Task 2.2.
3. Remember to save the document and **backup** your work! Storing your work in multiple locations will help ensure that you do not lose anything if one of your computers fails, or you lose your USB Key.

Note: This is another tasks you need to **submit to Doubtfire**. Check the assessment criteria for the important aspect your tutor will check.