Flappy Bird – YSJ Game Development Decision Day

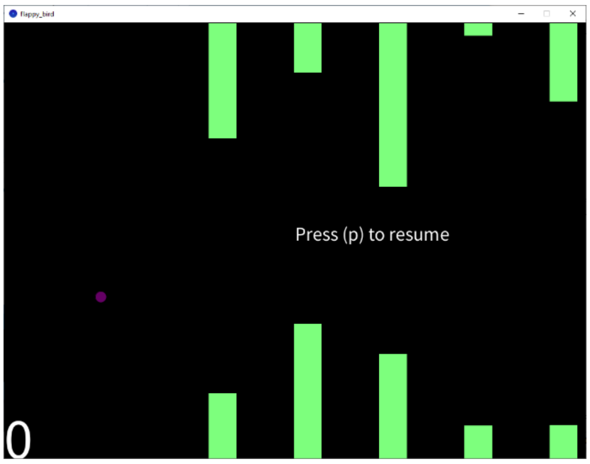
This activity will give you an introduction to programming a game. We're going to modify a clone of the game Flappy Bird (https://flappybird.net/).

Getting Started

You can find links to the files you need on the desktop. If you are doing this activity from home, you will need to locate the files you have downloaded. To get started, open the `Flappy Bird` folder and double click on `flappy-bird.pde` to open the Processing IDE (Integrated Development Environment).

Processing is a free programming language designed for creating art and animation. There is more information at the end of the document if you'd like to learn more.

Playing the Game



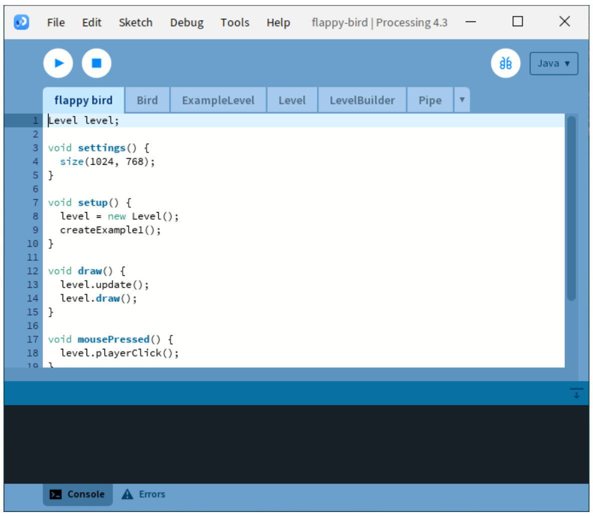
When you've got a program loaded in Processing you can run it simply by pressing the **Play** button at the top left of the Processing window.

The program is a simple clone of the game Flappy Bird. The goal of the game is to fly your bird as far as possible, avoiding the pipes. You have to keep the bird between the pipes by clicking the mouse button to make the bird flap upwards.

The graphics are a little simple in this version, but hopefully you get the idea.

Task: Play the game a few times until you're ready to move on.

Exploring the Source Code



```
Level level;

void settings() {
  size(1024, 768);
}

void setup() {
  level = new Level();
  createExample1();
}

void draw() {
  level.update();
  level.draw();
}

void mousePressed() {
  level.playerClick();
```

The text in the white area is the source code. The code is spread across several files. You can see the names of these files in the tabs across the top of the white text area. By clicking on one of the names tabs you can see (and edit) the contents of that file.

The *entry point* of the program (where the code starts running) is the file `flappy_bird`. This references *classes* and *methods* defined in other files.

Task: If you like, take a look at some of the following files to get an idea of what they contain. Or feel free to dive straight in to the activities below.

This is a brief overview of each source code file. Feel free to skip over this section.

flappy_bird

The *settings()* and *setup()* methods prepare the game. These methods are called by Processing automatically when the program starts.

The *draw()* method *updates()* the level information, then it *draws()* it. This method is called by Processing many times each second.

This file also tells the **level** code when the mouse button or the ‘p’ key is pressed.

Bird

A *Bird* is a moving circle controlled by the player. The “Bird” file is a *class*, which tells the program what the symbol “Bird” means and what information to store for each Bird, including:

- position on screen (x,y)
- velocity (xvelocity, yvelocity)
- colour

The code tells it how to *move()*, *checkForCollision()* with the pipes and *draw()* itself.

Level

A *Level* is a collection of pipes, gaps, and a finish line. The “Level” file is a *class*, which tells the program what the symbol “Level” means. It holds the information about all the pipes and the gaps between them, it will keep track of the bird in the level and your score.

It has code to *update()* and *draw()* the level, it also has code to make the bird flap when you click the mouse button – *playerClick()*.

Pipe

A *Pipe* is an obstacle that appears in the level. The “Pipe” file is a *class*, which tells the program what the symbol “Pipe” means and what information to store for each Pipe.

This file also defines how to *draw()* a pipe.

LevelBuilder

A *builder* is a software design pattern for helping to create objects easily.

This file contains methods that are used to create a level. A level needs *pipes* with *gaps* between them, a *finish* and, of course, a *bird* to play the level.

Example Level

This file contains a method that creates an example level of 100 random pipes.

Challenges

These challenges are designed as a gentle introduction to games programming in Processing. If you've programmed before, especially if you've programmed in Java, you may find the activities quite easy. Either way, they'll help familiarise you with some of the specifics of Processing.

Challenge 1: Change the colour of the bird

Our Flappy Bird is currently a red circle but we can make it any colour we like.

Open the **Bird** file by clicking on the Bird tab at the top of the white text area. This file contains all the information and code directly related to the bird.

At the top of the file, on line 11, is the line:

```
color colour = color(255, 0, 0);
```

This line creates a variable called **colour** that is used to set the colour of the bird.

On line 42 the command `fill(colour)` sets the current fill colour so that when the next line draws the bird circle it is filled with our selected colour.

How do we pick the colour? The code on line 11 sets the colour using three numbers. Each number represents an amount of red, green and blue in the colour. 0 means none of that colour, 255 is as much as possible. So the line in the code sets the colour to 255 parts red, 0 parts green and 0 parts blue, making it bright red.

Here's that line of code with the numbers coloured to match the colour they represent.

```
color colour = color(255, 0, 0);
```

We could make the bird green by changing the line to

```
color colour = color(0, 255, 0);
```

We could make the bird purple by changing the line to

```
color colour = color(100, 0, 100);
```

By lowering the number to 100 we've made the colour darker.

> **Task**: Try experimenting with the numbers to change the colour. Remember to save your changes and press the play button to see what you've done.

Challenge 2: Change the colour of the pipes

The pipes are coloured the same way the birds are but this time the code is in the Pipe file.

> **Task:** See if you can find how to change the colour of the pipes.

Challenge 3: Randomise the colour of the pipes

Line 16, 17 and 18 in Pipe set variables called red, green and blue to random numbers between 0 and 255.

If you look at line 20 you will see the code is all in gray. This is because it is commented out. If we put // at the start of a line Processing will ignore it. We do this to add comments to code so we understand what it is doing when we come back to it later.

Change line 20 from

```
    //this.colour = color(this.red,this.green,this.blue);
```

to

```
    this.colour = color(this.red,this.green,this.blue);
```

Save the file and run the game. You should see the pipes are now all different colours.

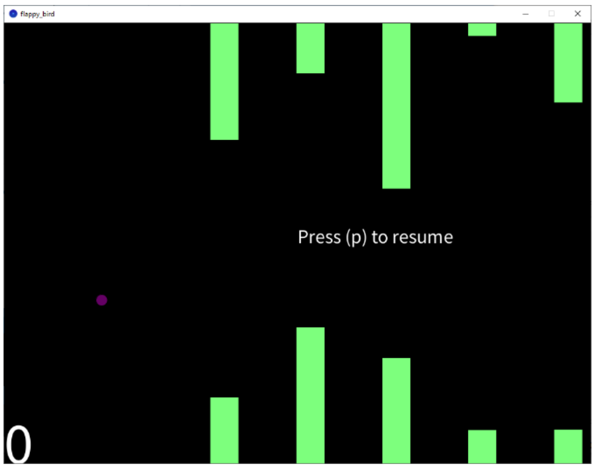
If you change lines 16, 17 and 18 to

```
    this.red = int(128+random(128));
    this.green = int(random(128));
    this.blue = int(random(128));
```

Save the file and run the game. You should see the pipes are still all random colours but with a tendency to reddish colours.

Task: Alter the code so that it generates pipe colours of your preference.

Challenge 4: Change the level



In this game level building is pretty simple.

Each level has a bird and some pipes. The pipes always come in pairs, one coming down from the top of the screen and one going up from the bottom of the screen, this pair has a space between them for you to fly through. Between one pair of pipes and the next is a gap.

The code to make levels is in LevelBuilder.pde. This file has four functions in it:

- addBird() – adds a bird to the start of the level
- addPipe() – adds a pair of pipes to the level
- addGap(width) – adds a gap, width wide, after the last thing added to the level
- addFinish() – adds a finish line

Let's change the level to a different example level. In *flappy-bird.pde*, change line 9 from

```
    createExample1();
```

to

```
    createExample2();
```

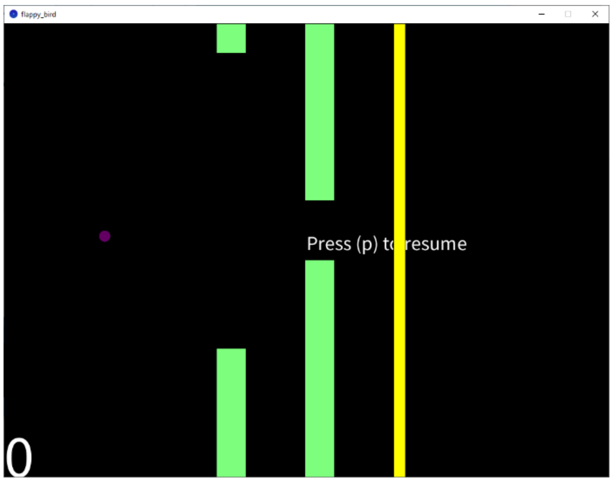
Task: Play example level 1 and example level 2. Compare the code in `createExample1()` and `createExample2()`. Example 2 is easier to understand, but the level is very short. The code for Example 1 is more complex, but it generates a much larger level with the same number of lines of code

Challenge 5: Create your own level

To make an interesting and fun level we need to change the space between the pipes and the height of the gap.

If you look at LevelBuilder you'll see there are two different `addPipe()` methods. One has empty brackets () and the other has brackets containing *parameters* `(int gapOffset, int gapSize)`. We use parameters to pass data into a method. We will use the `addPipe(int gapOffset, int gapSize)` version to tell the method how far down from the top of the screen we want the space between the pipes to start and how big we want the space to be.

If we use `addPipe(50, 500)` we'll get a pair of pipes on the screen with a space that starts 50 pixels down from the top of the screen and is 500 pixels long before the bottom pip starts.



If we used `addPipe(300, 100)` we'll get a pair of pipes on the screen with a space that starts 300 pixels down from the top of the screen and is 100 pixels long before the bottom pip starts. It'll be a lower, smaller gap.

> **Task:** Edit the code in `createExample2()`. For each `addPipe()` method call, add `gapOffset` and `gapSize` parameters.

Challenge 6: Behave like a Computer Scientist

There are many ways we can analyse a program. One is to study the source code and work out what it should do. This is the 'mathematical' approach. For example, you could prove that the following program always returns `true`:

```
return 1 + 1 == 2
```

However, because most computer programs are very complicated, we cannot understand them fully from the code alone. In order to understand their behaviour, we have to behave like scientists, testing different ways to program something and comparing their effectiveness.

> **Task:** Experiment with changing the code that makes the level. Change the size of the gaps between pipes and add some new pipes. Try adding a small gap after the bird and then another bird to see what happens. What happens if you call `addFinish()` more than once? What can you learn about how the program works?

Challenge 7: Taking It Further

If you have some experience with programming, you will be able to make more significant changes to the code. The code in the `createExample1()` method shows how to create a random level using a **for loop** and a **random number generator**. Those two things open up a lot of possibilities for creating levels with code.

This line means anything contained in the braces {} that follow it will be repeated 100 times:

```
for (int i=0;i<100;i++)
```

This line randomly generates the size of a gap based on the height of the screen, which is used in the subsequent line to create a pipe:

```
int gap = (int) random(250, height - 50);

addPipe((int) random(10, height-gap -10), gap);
```

Task: Start by altering the values in these lines and seeing what happens, or dive straight in to writing your own loops to generate a level. See what you can achieve.

Processing

"Processing is a flexible software sketchbook and a language for learning how to code. Since 2001, Processing has promoted software literacy within the visual arts and visual literacy within technology." – Processing Website, https://processing.org/

Processing is a free software package and programming language designed to make learning to program quick and easy. It is available free for most platforms at https://processing.org/.

A number of tutorials are available on the Processing website - https://processing.org/tutorials.

Full reference documentation is available too - https://processing.org/reference.