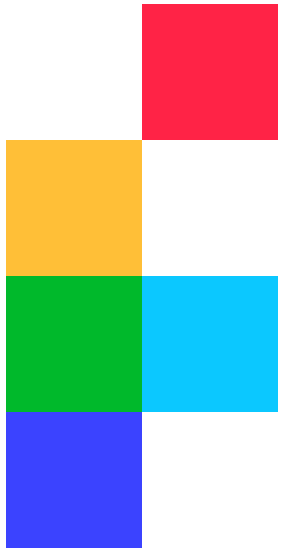
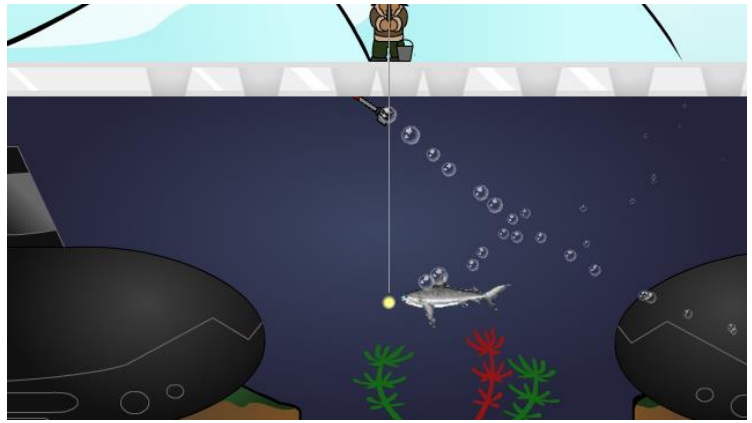


ActionScript 3
w/ Flixel Library

What our club has made with Flixel so far:

(HaxeFlixel)



Features:

- Many annoying things that are often used in making a game already implemented nicely so it becomes handy for you when you need them (they may be not hard to implement but you kindda don't want to have to code it every single time). E.g.
  - Game Pausing
  - Camera
  - Screen Shake
  - Record & Replay
  - Game Save
  - Pathfinding
  - …
- Aimed for making pixel-styled games; pixelize everything in stage
  - But you can still use it for making non-pixel-styled games
- Good optimization, easy object recycling, etc.

Downsides:

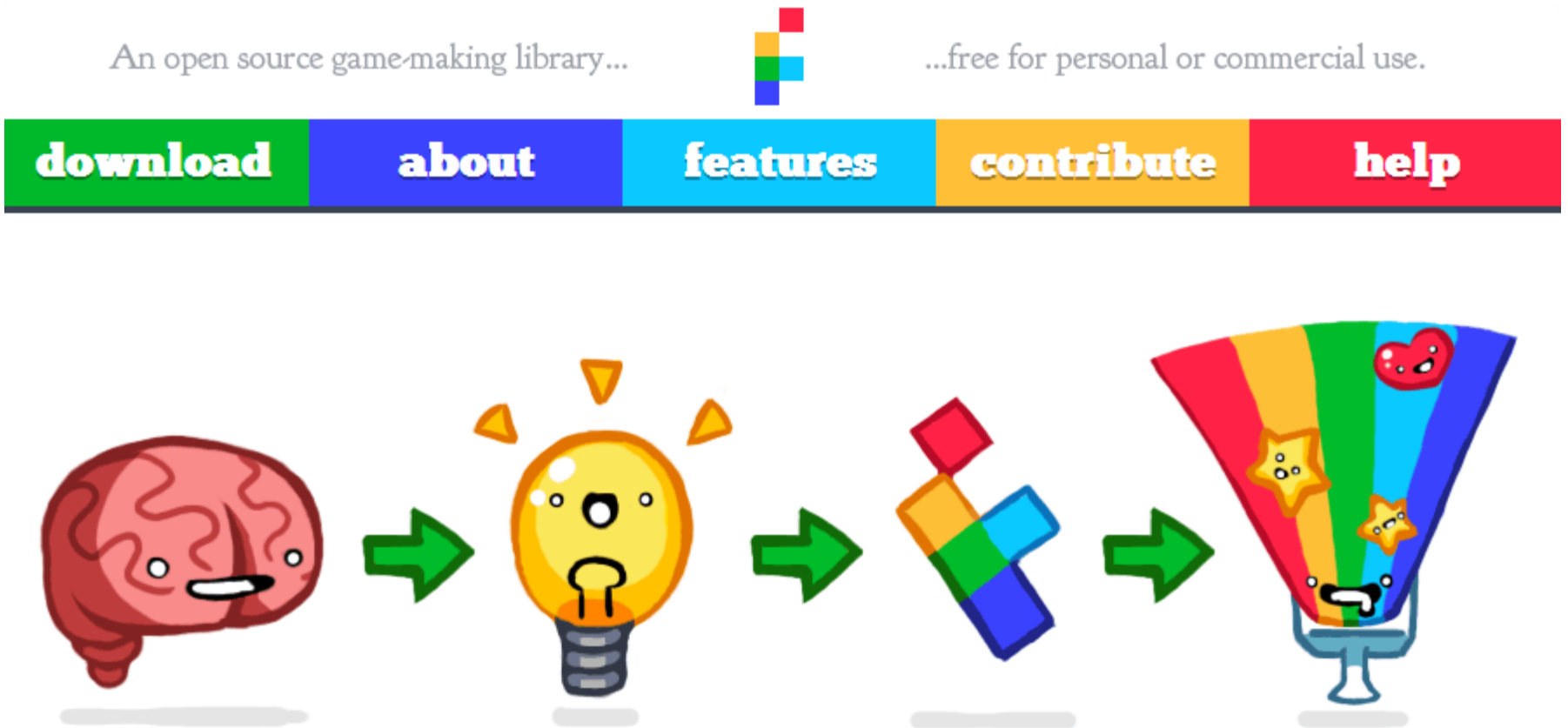
- Often need some hardcoding. But, I mean, it's ActionScript 3 anyway.

First of all…

You need FlashDevelop

Download: http://www.flashdevelop.org/

Get Flixel Library: http://flixel.org/download.html

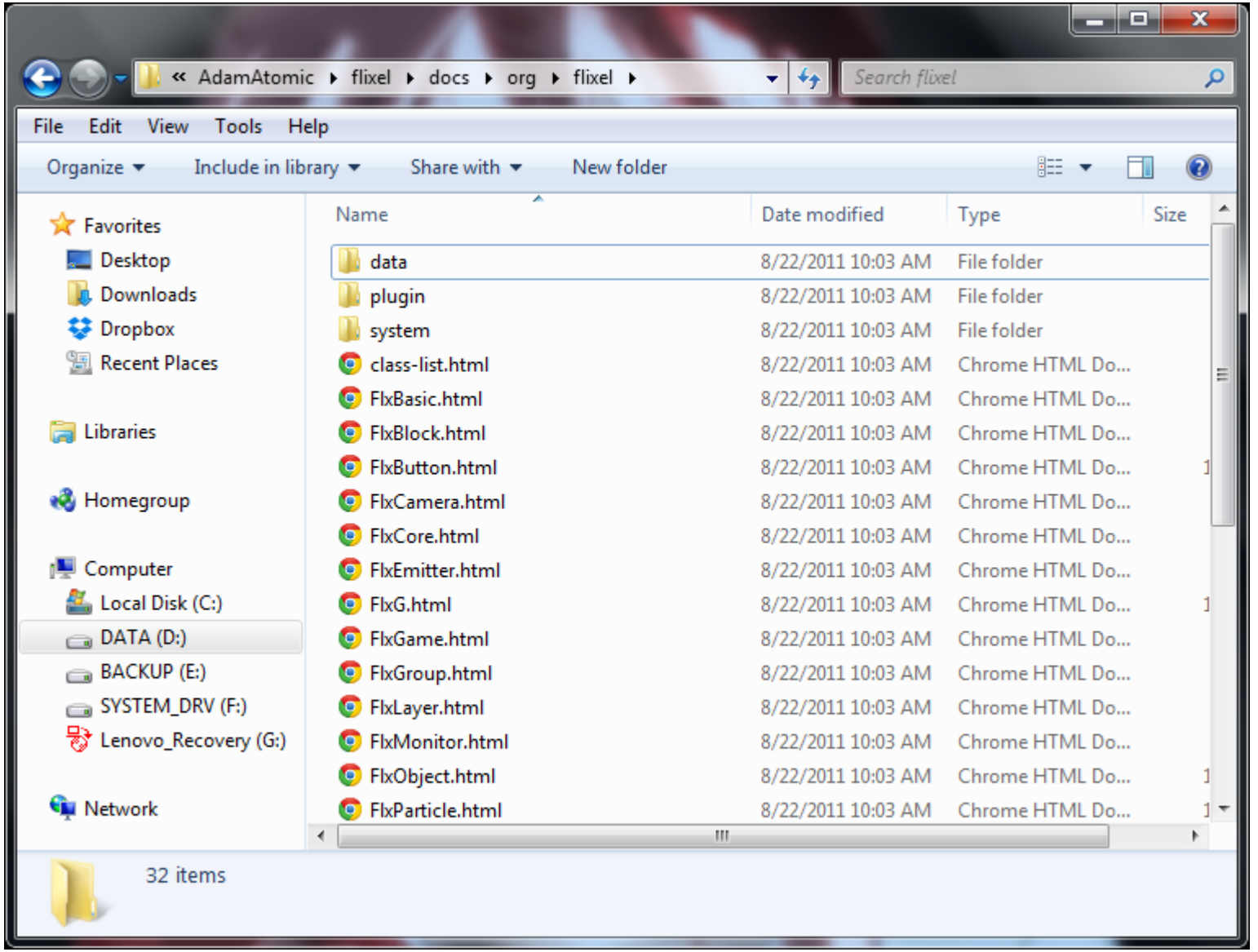


Flixel is an open source game-making library that is completely free for personal or commercial use. Written entirely in Actionscript 3, and designed to be used with free development tools, Flixel is easy to learn, extend and customize.

Documentation:

It comes with the library:
..\AdamAtomic\flixel\docs\org\flixel

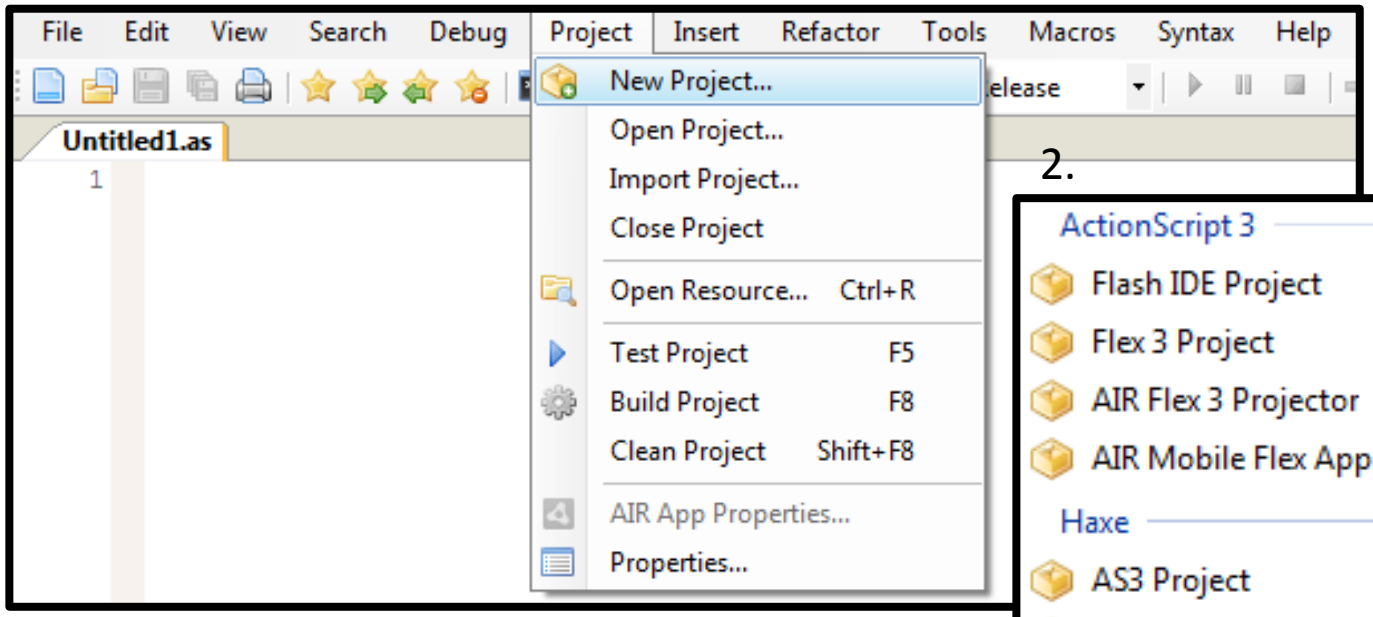
Plus you can find it online, too.



Getting Started:

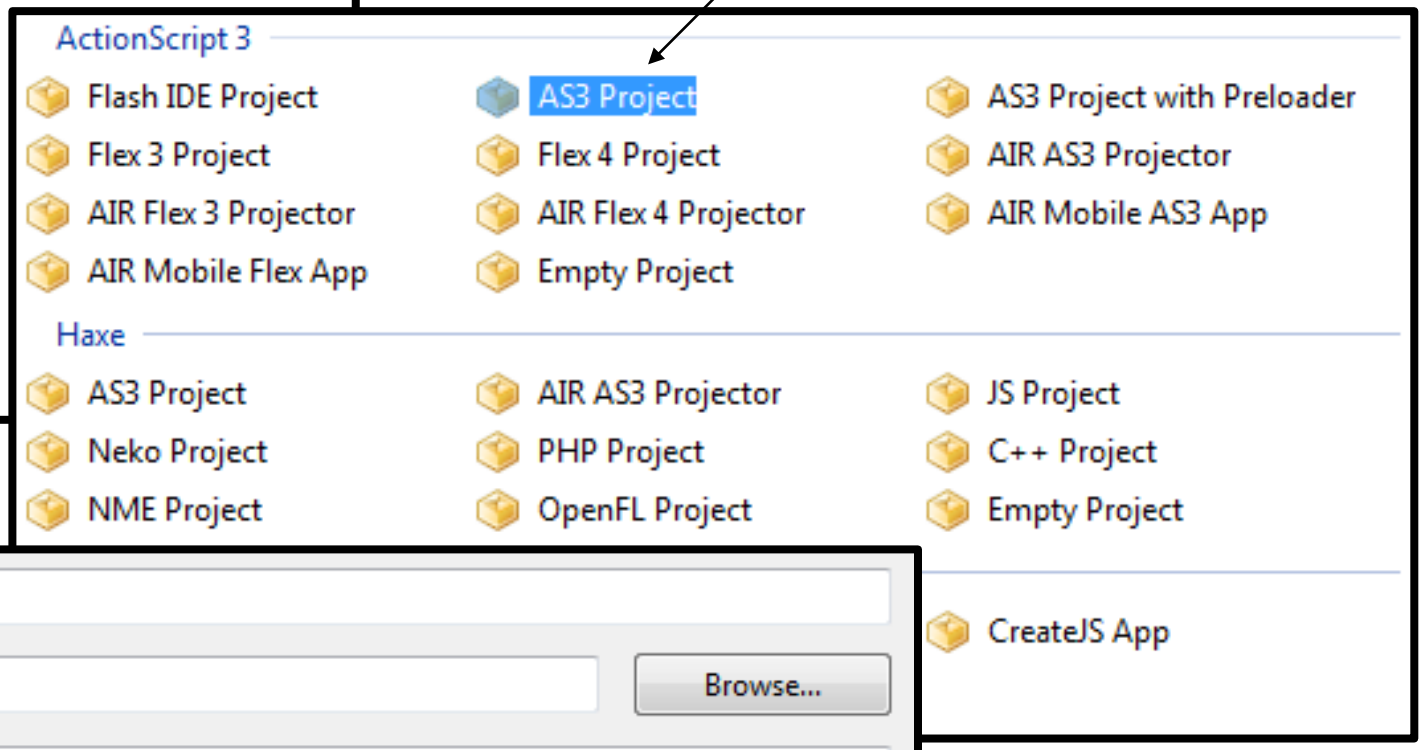
Make a new project!

1.

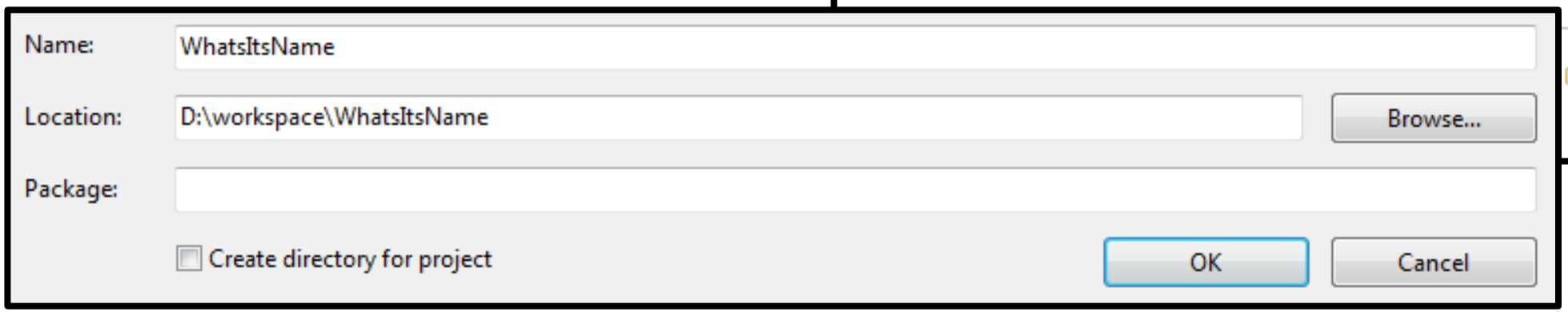


Just select a AS3 Project

2.



3.



Copy-n-Paste the flixel lib into your project folder or build path.

Then change the main into a flixel game:

Before

Main.as*

```
1  package
2  {
3      import flash.display.Sprite;
4      import flash.events.Event;
5
6      /**
7       * ...
8       * @author Wenrui Wu
9       */
10     public class Main extends Sprite
11     {
12
13         public function Main():void
14         {
15             if (stage) init();
16             else addEventListener(Event.ADDED_TO_STAGE, init);
17         }
18
19         private function init(e:Event = null):void
20         {
21             removeEventListener(Event.ADDED_TO_STAGE, init);
22             // entry point
23         }
24
25     }
26
27 }
```

After

```
package
{
    import org.flixel.FlxGame;

    [SWF(backgroundColor = "#000000", frameRate = "60", width = "480", height = "480")]
    [Frame(factoryClass="Preloader")]

    public class Main extends FlxGame {

        public function Main():void {
            super(480, 480, PonGame);
        }
    }

}
```

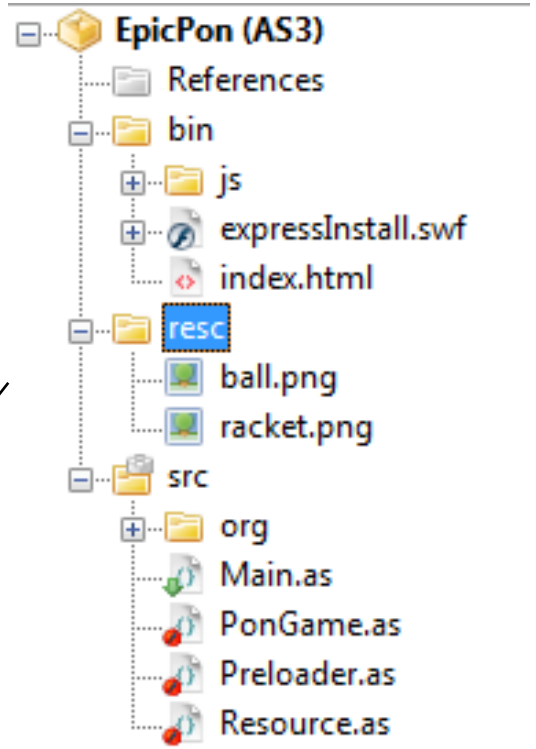
What we need:

- Flixel Preloader → Preloader.as
- Game file → WhateverYouWantToCall.as → In this case, “PonGame.as”
- A file to call all the resources → Resource.as

Preloader.as

```
package{
        import org.flixel.system.*;

    public class Preloader extends FlxPreloader
    {
        public function Preloader():void
        {
            className = "Main";
            super();
        }
    }
}
```



MS-Paint and organize your material in a resource folder

Resource.as

```
package  {

    public class Resource {
        [Embed( source = "../resc/racket.png" )] public static var IMPORT_RACKET:Class;
        [Embed( source = "../resc/ball.png" )] public static var IMPORT_BALL:Class;
    }
}
```

Basic structure of a game file (or any other class that extends FlxState)
Two key functions: create() and update()

```
package
{
    import org.flixel.*;

    public class PonGame extends FlxState
    {
        public override function create():void {
            super.create();
        }

        public override function update():void {
            super.update();
        }
    }

}
```

You can now run it already!

Initialize/Create all the game assets before create()

```
public static const WID:Number = 480;
public static const HEI:Number = 480;

public var player1:FlxSprite = new FlxSprite();
public var player2:FlxSprite = new FlxSprite();

public var balls:FlxGroup = new FlxGroup();
```

Must-know for the sake of hardcoding

Three steps of drawing and loading a flixel sprite into the stage:

```
player1.loadGraphic(Resource.IMPORT_RACKET);
player1.set_position(60 - player1.width, (HEI - player1.height) / 2);
this.add(player1);
```

← Position = Left upper corner of the sprite

Equivalent to addChild(…)

Note: loadGraphic will determine the width and height of the sprite automatically unless otherwise stated

About balls – should we just make FlxSprite again or make a class that extends FlxSprite?

We'll see!

Think about our update cycle.

```
public override function update():void {
    super.update();

    if (!gameover) {
        update_control();
        update_balls();
    }
}
```

I'd like to store a ball's vx, vy and update its position by calling ball.update_position() so I will probably make a class that extends FlxSprite

Building a class that extends FlxSprite is just like building a normal class (syntax pretty similar to java programming, too) and you can also choose to write your own customized update function instead of directly overriding update().

```
public function Ball(x:Number = 0, y:Number = 0, vx:Number = 0, vy:Number = 0) {
    this.vx = vx;
    this.vy = vy;

    this.loadGraphic(Resource.IMPORT_BALL);
    this.set_position(x, y);
}

public function update_position():void {
    this.x += vx;
    this.y += vy;
}
```

What else should we do to handle situations in this game properly?

Suggestion: another function that will probably become handy:
should_kill():Boolean

```
var ball:Ball = new Ball(player1.x + player1.width,
                         player1.y + player1.height / 2,
                         random_float(3,5),
                         random_float( -4, 4));
balls.add(ball);
this.add(balls);
```

You can just add it to FlxGroup as long as the Ball class extends FlxSprite so it is considered as a Flixel Object.
Remember to add the whole group to the stage.

```
// player 1 control
if (FlxG.keys.pressed("W")) {
    player1.y -= MV_SPD;
    if (player1.y < 0) {
        player1.y = 0;
    }
} else if (FlxG.keys.pressed("S")) {
    player1.y += MV_SPD;
    if (player1.y > HEI - player1.height) {
        player1.y = HEI - player1.height;
    }
}
```

```
// handy collision test
FlxG.collide(player1, balls, function(player:FlxSprite, ball:Ball) {
    ball.bounceX();
});
FlxG.collide(player2, balls, function(player:FlxSprite, ball:Ball) {
    ball.bounceX();
});
```

Handy FlxG static functions that does lots of cool stuff!
E.g. keyboard listener, collision test, camera handling, etc.

How do we make it more epic?

1. Gameplaywise
    - Suggestions:
        - Spawn more balls
        - Making irregular trajectory (mess up with update_position() in Ball)
2. Special Effect
    - Suggestions:
        - Screen shake when gameover
3. Sound
    - Just use bfxr to generate; and FlxG handles sound effect
    - Adding sound to Resource file is similar to that of adding an image material
4. Other
    - Auto start-over
    - Kill Cam (?) → FlxG does have game recording handling, but has a lot of limits and tricks
    - …

Trick of auto start-over

```
} else {    // gameover
    if (ct_wait == 0) {
        FlxG.camera.shake();
    } else if (ct_wait >= 180) {
        FlxG.switchState(new PonGame());
    }
    ct_wait++;
}
```

(ct_wait is the timer)

FlxG has a nice switchState() function
Usually, this function is used to switch between game menu, game's main file, endgame sequence, etc.
(You make different State with each of them)
However, there is a trick – you can switch to yourself, too! And everything will start-over

Then how do you keep track of scores?
CSE 142, 143 will probably tell you to pass parameters, but here's a trick – make use of static variables!

```
public static var score1:int = 0;
public static var score2:int = 0;
public var sc1:FlxText;
public var sc2:FlxText;
```

There’s our cool Pon game!