Tutorial – Splitting a Project

In this week's lesson you learnt how to break up your game into objects, and how the definition of each object should be contained in its own file.

In the tutorial for this week, we're going to start with a fresh project and add two new objects – a player object and the keyboard object.

The player object will hold the definition of our player – eventually our player will contain an animated sprite and some code to jump around our level, but for now it will just contain some simple update logic and draw a static sprite.

The keyboard object will handle key press events. Its functionality will be very similar to what we did in the Asteroids game, but by making a keyboard object we can contain the keyboard handling code into one file and simplify how key press events are handled by our game.

Once all that is done, we'll create a GIT Hub repository for you to save your work to.

The Keyboard Object:

Since the keyboard object will have no dependencies, we'll start by defining this object.

1. Create a new JavaScript file, called keyboard.js
2. Insert the following code into this file:

```javascript
var Keyboard = function() {
    var self = this;

    window.addEventListener('keydown', function(evt) { self.onKeyDown(evt); }, false);
    window.addEventListener('keyup', function(evt) { self.onKeyUp(evt); }, false);

    this.keyListeners = new Array();
    this.keys = new Array();

    // Key constants. Go here for a list of key codes:
    // https://developer.mozilla.org/en-US/docs/DOM/KeyboardEvent
    this.KEY_SPACE = 32;
    this.KEY_LEFT = 37;
    this.KEY_UP = 38;
    this.KEY_RIGHT = 39;
    this.KEY_DOWN = 40;

    this.KEY_A = 65;
    this.KEY_D = 68;
    this.KEY_S = 83;
    this.KEY_W = 87;
    this.KEY_SHIFT = 16;
};
```

```
Keyboard.prototype.onKeyDown = function(evt)
{
        this.keys[evt.keyCode] = true;
};

Keyboard.prototype.onKeyUp = function(evt)
{
        this.keys[evt.keyCode] = false;
};

Keyboard.prototype.isKeyDown = function(keyCode)
{
        return this.keys[keyCode];
};
```

Our keyboard class is relatively simple.

When the onKeyDown event occurs, we put the value 'true' in the keys array at the position indicated by the key's numeric keyCode identifier. When the onKeyUp event occurs, we do the same thing, this time setting the value to false.

To tell whether a key is currently pressed or not, all we need to do is query the value in the array for a specified key code.

For example, to know whether or not the space bar is currently being pressed, we would call isKeyDown(KEY_SPACE), or isKeyDown(32)

This is a very simple method of keyboard event handling, and while it is not very optimized it is sufficient for our needs.

The Player Object:

The slides for this week's lecture contain a simple player definition. We'll use the code presented in these slides as the base for our Player object.

Create a new file for your player object, called player.js, and copy the code from the code for the player object from the slides into this file (it will be the slide with that includes the player update and draw functions).

Modify the update function to make use of the keyboard object as follows:

```
Player.prototype.update = function(deltaTime)
{
      if( typeof(this.rotation) == "undefined" )
            this.rotation = 0;          // hang on, where did this variable come from!

      if(keyboard.isKeyDown(keyboard.KEY_SPACE) == true)
      {
            this.rotation -= deltaTime;
      }
      else
      {
            this.rotation += deltaTime;
      }
}
```

Update the Template:

To make things easier for you when starting a new project, we've created a template for you to use.

This template links the main.js file with the HTML file, creates a reference to the canvas, and contains a simple game loop for you to modify. It should take the pain out of starting a new game.

Download the template from AIE Portal now, extract it to a folder somewhere on your machine, and add the player.js and keyboard.js files we created above to this folder.

To get everything working we need to update the index.html file (so both the keyboard.js and player.js files are loaded) and the main.js file (to create the player and keyboard objects, and to update/draw the player object each frame).

1. Modify index.html as follows:

```
<html>
  <body>
    <canvas id="gameCanvas" width="640" height="480">
        This text is displayed if your browser does not support HTML5.
    </canvas>
      <script src="keyboard.js"></script>
    <script src="player.js"></script>
    <script src="main.js"></script>
  </body>
</html>
```

2. Modify the main.js as follows:

```
var player = new Player();
var keyboard = new Keyboard();

function run()
{
        context.fillStyle = "#ccc";
        context.fillRect(0, 0, canvas.width, canvas.height);

        var deltaTime = getDeltaTime();

        player.update(deltaTime);
        player.draw();

                // update the frame counter
        fpsTime += deltaTime;
        fpsCount++;
        if(fpsTime >= 1)
        {
                fpsTime -= 1;
                fps = fpsCount;
                fpsCount = 0;
        }

        // draw the FPS
        context.fillStyle = "#f00";
        context.font="14px Arial";
        context.fillText("FPS: " + fps, 5, 20, 100);
}
```

3. Run the project. You should see Chuck Norris slowly rotating in the centre of the screen. When you hold the space bar Chuck will start rotating in the opposite direction.

   If you don't see this, go back over the steps above to make sure you didn't miss anything. Make sure you copied the complete Player object code from the lecture slides.

Creating a GIT project:

We are going to use GitHub to store backups of our source code and resource files. This is a really really good idea because it means you an access your code from anywhere, and if you ever have disaster strike your computer then you have a back-up of all your hard work.

1. Go to https://github.com/ and create an account (and log into your account)
2. Select '**Create a repository**' from the list of links under that big image – just below the text that reads "Welcome to GitHub! What's next?"

3. Enter a repository name. It can be anything you like, but I recommend using the same name as your game (you have a name picked out for your platformer game, right?)
4. Fill in the other details, making sure you check the box for initializing with a readme file. (This just makes sure you don't end up with an empty project)

Unless you've paid money for GitHub you will have to make you're repository public. That's OK, the internet is a big place and nobody really wants to steal your game anyway. (Seriously, don't worry about it).

5. Push the 'Create repository' button and… wham, your repository is created.
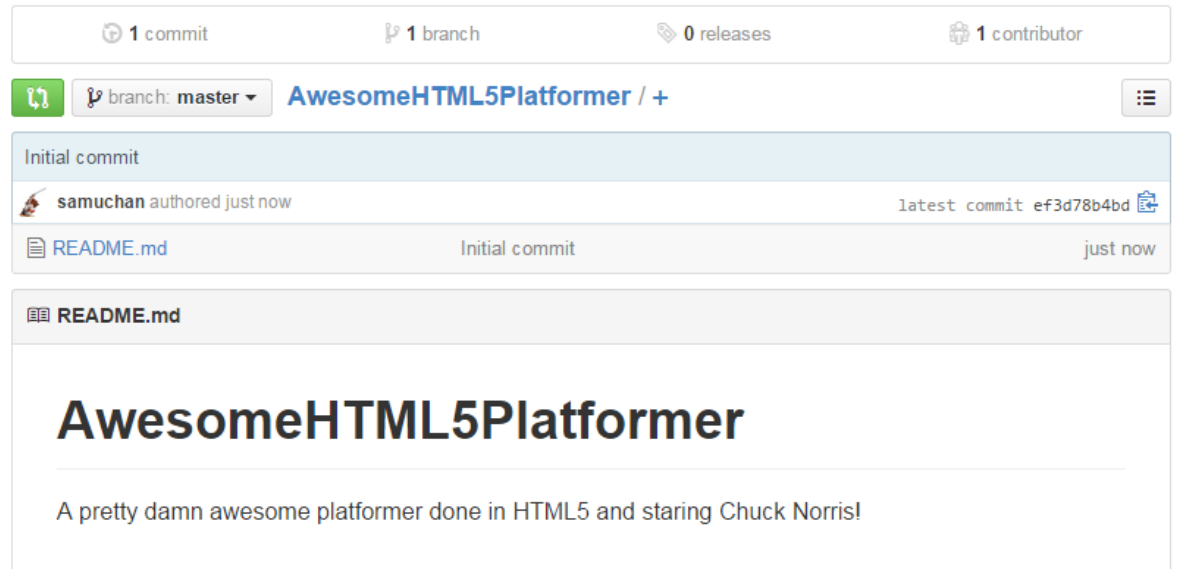
6. Download and install the GitHub client software by going to the appropriate site for your operating system
    - For Windows: https://windows.github.com/
    - For Mac: https://mac.github.com/
7. Launch the client program. Log in using your GitHub username and password. This will allow the client software to find your repositories
8. Press the '**+**' icon in the top-left corner of the application
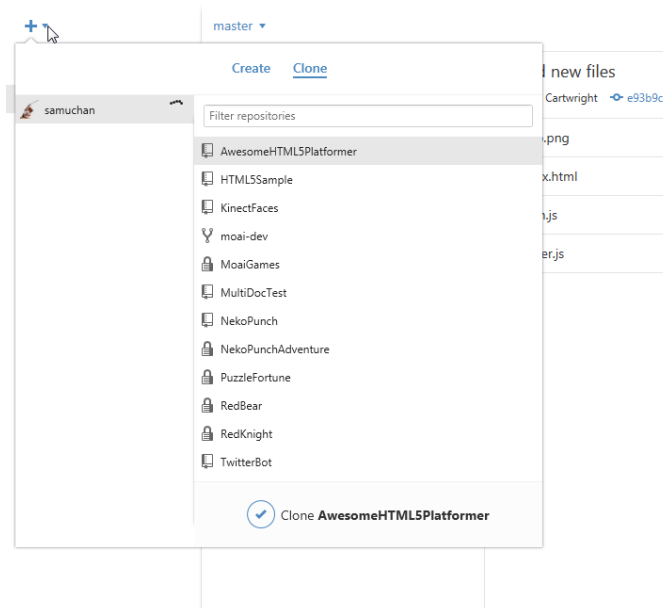
Click 'Clone' and select your repository from the list. Once you're done, click the tick icon.

9. Select a folder on your machine where the files should be downloaded to. GitHub remembers this location, so if you ever move this folder then GitHub won't be able to find your files and you'll need to set the project up again. (So choose wisely)
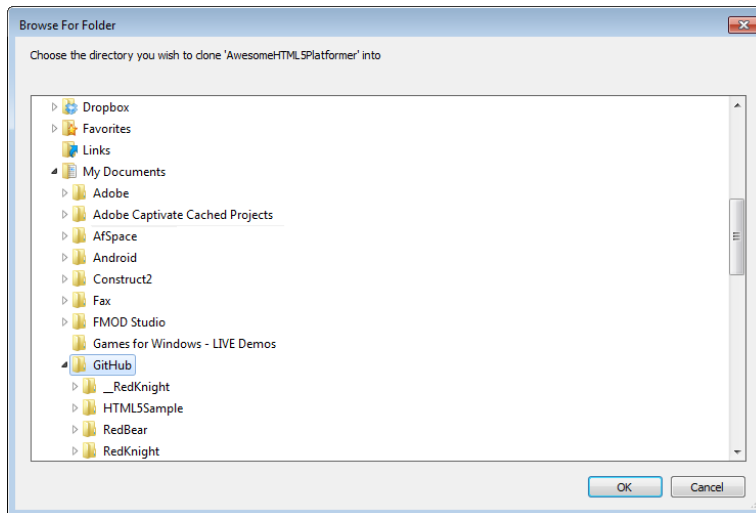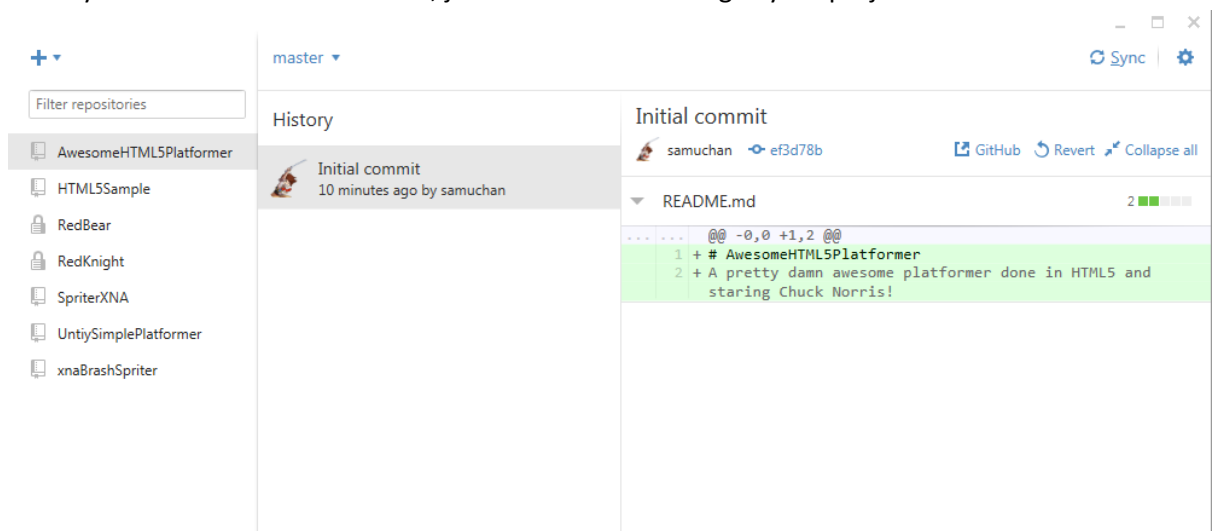

10. Files will download and directories will be made

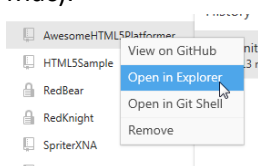Initially there won't be much to see, just a readme file sitting in your project.
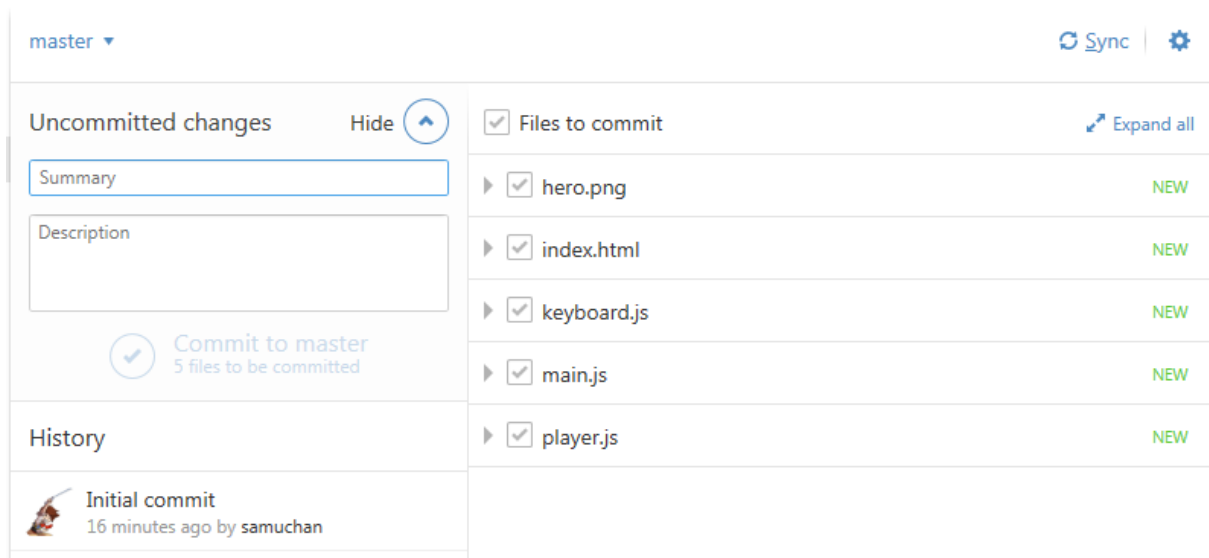
11. Open the folder where your project downloaded to. Either navigate there yourself, or right-click on your project name and select 'Open in Explorer' (or 'Open in Finder' if you're on a Mac).

12. Copy your files from this tutorial into the repository folder.

If you click on the '**Show**' button next to the 'Uncommitted Changes' text you should see the files you just copied to the repository folder
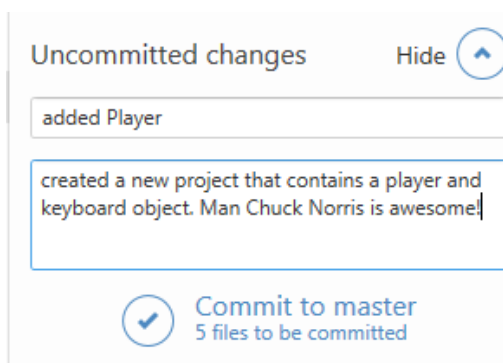

Uncommitted changes mean changes that you have made (like copying in new files) but haven't uploaded to the server yet.

13. Enter a summary for your change (something like "added Player") and a description ("created a new project that contains a player and keyboard object. Man Chuck Norris is awesome!")

It's a really good idea to always put in a descriptive summary and description because if you ever need to go back to a previous version you'll know exactly what is different.

After you've entered text into the Summary and Description fields you'll be able to press the tick icon. Press that button now to commit your changes.

14. Your changes have now been committed. **BUT** they're still only on your machine.

This is one of the 'features' of Git, but it can be a little confusing. Committing a change means that Git remembers what changes you just made and will allow you to go back to a previous version if you want. But your changes haven't been uploaded to the server yet.

To upload to the server, you need to Sync. This is an additional step because Git knows that sometimes you'll want to work offline.

15. To Sync your changes to the server (i.e., to upload your files), press the '**Sync**' button in the top-right corner of the application
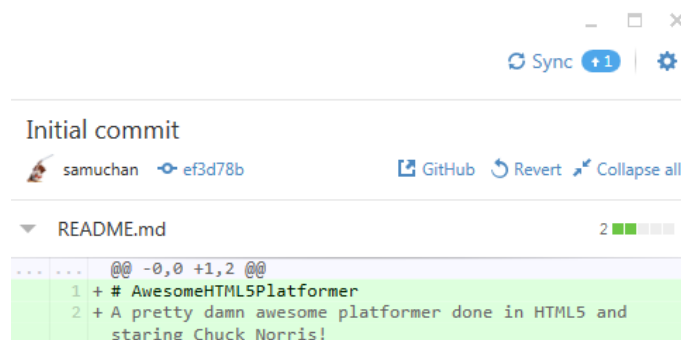
Your files will now upload to the GitHub site. Afterwards your client will display the 'No Local Changes' message.

When you return to the Git client, you'll see that your history has been updated with the change you just made.
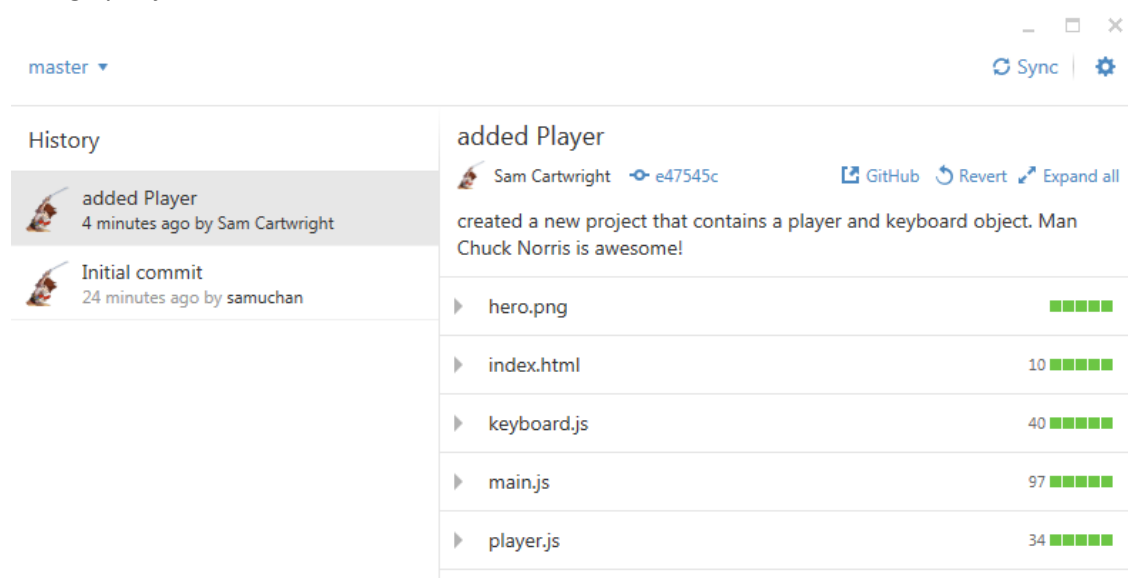
16. If you want to confirm your changes uploaded to the server, refresh the GitHub web page and you should see all your files.

HALP!:

If you ever need help with your assignment (and you are studying via online), you can now upload your code to GitHub and send your teacher the GitHub URL for your project (via the helponline@aie.edu.au email address).

The GitHub URL should look something like this:
https://github.com/samuchan/AwesomeHTML5Platformer

Exercises:

1. Up until now whenever we wanted to store a 2D vector we'd create an x variable and a separate y variable. This bloated our code out a little

   Use your knowledge of JavaScript objects to create a Vector2 object. This object should store the x and y variables and be written in a file called vector2.js.

2. Add the following functions to your Vector2 object:
   - set(x, y)   - pass in an x and a y argument to allow setting both properties at the same time
   - normalize()   - normalizes the vector
   - add(v2)   - add the input 2D vector to this vector
   - subtract(v2)   - subtract the input 2D vector from this vector
   - multiplyScalar(num)   - multiply this vector by the input number

3. Add an Enemy object (copy the code from the Player object if you wish). Use whatever image you like for the enemy.

CHALLENGE

4. Add a Bullet object. Copy your collision detection function from your Asteroids game and see if you can modify your code so that your player can shoot a bullet at your enemy (and the enemy dies).