Aston University
Birmingham

Tutorial 1

Introduction to the course:

In this tutorial you will be using Javascript with our wrapper API to get a basic game going. The steps you will be completing are:

- Setting up the canvas
- Adding Mario
- Moving mario around

Open up the tutorial1.html file in Firefox, it must be Firefox or it may not work.

If you open up the tutorial1.js file in the js folder you will be able to start coding for the tutorial.

Before we begin there are a few things that you need to know about coding in javascript.

One of the most basic parts of any programming language is a variable, a variable is used to store something, it could be a string of letters, it could be a number it could be anything! A variable in javascript is created with the *var* keyword and then the name of the variable is given after it. To assign a value to a variable you use the equals operator. So if you wanted to create a variable to store your name you would do this as follows:

```
var name = "josh";
```

You may have noticed that there are quote marks around my name, any string in javascript must be enclosed in quotes as it allows it to be understood as a string. If you dont do this, it will be thought to be something else such as another variable which could cause some errors.

FYI; quotes can either be singular(‘ ‘) or double(“ “) it does not matter as long as the same are used with each other. This means that double and single(“ ‘) is not valid.

You may also notice that I put a semicolon at the end of the string, this is used as a way of ending the variable, this tells the system that there is nothing left to add to the variable.

The next part to javascript is functions, functions allow you to create a block of code whenever you chose. If you look in the tutorial1.js file (all of the .js files are in the js folder) you will see three functions,

preload, create and *update*.

preload is where you load in any assets before the game loads up, this is usually any images that will be used in your game.

create is where you physically have the game make the objects to be used in the game so this could be your main character for example.

update is where most of your game logic will go, this function is called every frame. Your game could run at 30 frames per second this means that it will update 30 times a second.

The final part is object, objects allow you to store groups of data together, our wrapper api contains many objects that allow you to put your game together. Examples include the game object, the Player object and many more!

Step 1

Open up *tutorial1.js* in a text editor such as gedit(linux) or notepad++(windows), you will notice the three functions mentioned above. Your first task is to create a game object which is the centre of your game. Firstly you will need to create a variable called game, you will want to put this variable in the section marked global variables as you want it to be accessible through the file.

You should now have your game variable at the top of the file the next step is to assign something to it. This is where object creation comes in, using the *new* keyword you can create a new object of a given type.

To create a Game object you will need to use the new parameter and pass through some information that will be assigned to that object. The Game object takes three parameters:

1. *width*: an integer value specifying the width of the game in pixels, if passed through as a string (enclosed in quotes) and smaller than or equal to 100 then it will be determined as a percentage.
2. *height*: an integer value specifying the height of the game in pixels, if passed through as a string (enclosed in quotes) and smaller than or equal to 100 then it will be determined as a percentage.
3. *name*: this is the name of your game

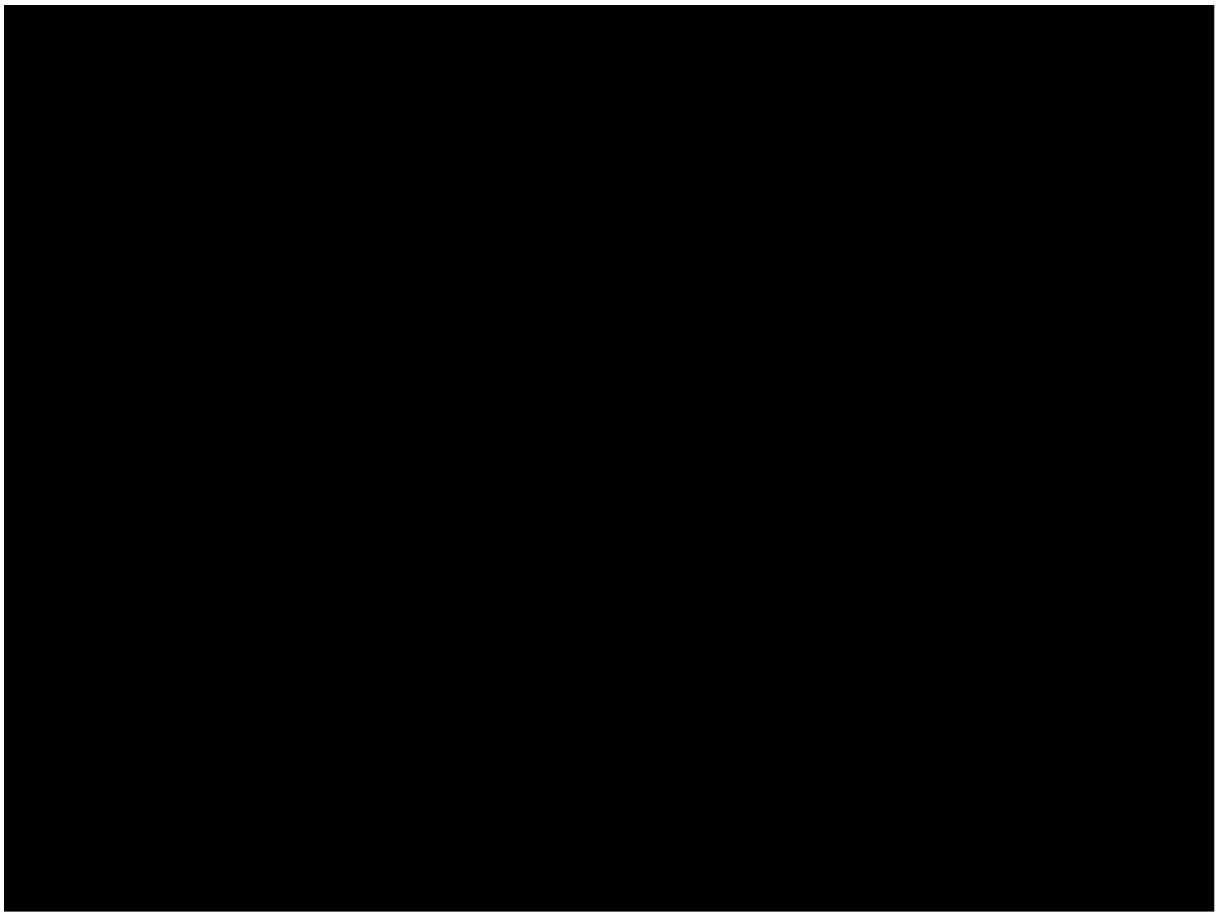
So to assign a Game object to your variable game you will do something similar to:

var game = new Game(800, 600, “Tutorial 1”);

Note: Be careful if you copy in the line above, quotes may not be encoded correctly causing errors

Make sure you put this just below your global variables and outside of any functions as it has to be the first thing created!

Open up Tutorial1.html in Firefox, you should have something similar to the picture below, if not ensure you have passed through your parameters in the correct order.



A black screen! We have the game area created, now to add stuff to it!

Step 2

The next part of your game is your main player, they can be represented with a Player object, you will first need to create a variable to store your Player. Go ahead and create one underneath where you created your game object and call it player.

You will then need to create a Player object to assign to your player variable. The player object takes three parameters:

1. *startX* : This is the x coordinate for the starting position of your player
2. *startY* : This is the y coordinate for the starting position of your player
3. *game* : This is your game object that you created earlier
4. *spriteImage* : This is the image that you shall be giving your player
5. *spriteX* : This is the height of your image
6. *spriteY* : This is the width of your image

Create a new Player object and assign it to the player variable as you did before with the Game object. This time you will want to create the object within the *preload* function this time. For the tutorial your *spriteImage* will be "img/mario-sprite.png" and you *spriteX* will be 17 and your *spriteY* will be 32 as mario is17X32 pixels. You can set your *startX* and your *startY* to whatever you like, just ensure it is in bounds.

So you should have something like:

```
player = new Player(50,50, game, "img/mario-sprite.png", 17, 32);
```

This sets the player up at the position 50,50, if you reload tutorial1.html you will see a black screen, why is this? We have not created the image yet! All that the above code did was load the image into the game.

To actually create the image we use the *createSprite* function from the Player class which creates the loaded sprite and adds it to our player. This function takes no parameters and must be placed inside the *create* function. As it is a function of the player class we need to use our player variable to invoke it, to do this we use a full stop between the variable and the function. So it will look something like:

```
player.createSprite();
```

Now if you refresh your *Tutorial1.html* file in Firefox you should see Mario!

Step 3 : Adding movement!

The api allows you to use keys on your keyboard as input meaning that you are able to control your character with the directional keys!

The first thing that you need to do is create 3 variables, one for a Keys object, this object stores all of the keys that can be used. You then need a variable for your left key and one for your right key. Create 3 global variables with the names *keys*, *left* and *right*

```
var keys;
var left;
var right;
```

Now create a new *Keys* object in the *create* function and then use that object to assign values to the left and right variable:

```
keys = new Keys(game);
left = keys.createLeftKey();
right = keys.createRightKey();
```

Take a look at the Keys object here (http://eas-fyphost-02.aston.ac.uk/~smalljh/docs/Keys.html).

Also have a look at the Key object here (http://eas-fyphost-02.aston.ac.uk/~smalljh/docs/Key.html)

Now to get your keys to work you will need figure out when they are being pressed, each key that you just created has a nice function that will tell you this. You'll notice that the Key object

has a function *isDown* that will return a boolean value letting you know whether the key is being pressed down.

A boolean value can either be true or false. To test whether a key is pressed down you will need to use an *if statement*. If statements are made up of three different conditional statements:

1. *if* : specify a block of code to be executed if a condition is true
2. *else if* : specify another block of code to execute if that condition is true and the first one is not
3. *else* : a block of code to be executed if all of the other conditions are not true

The format would look something like this:

```
if(statement) {


} else if(statement2) {


} else {


}
```

If statements must have an *if* part, they can have any number of *else if* parts and can have 0 or 1 *else* parts. So in our case we will want to have an if statement that checks if the left button is being pressed and then if not if the right arrow is being pressed.

We want to check if the button is being pressed down on every update so make sure you place your if statement inside the *update* function.

The statement for each would be *left.isDown()* for the left key and *right.isDown()* for the right key. Your if statement would look something like this:

```
if(left.isDown()) {
        //move left
} else if(????) {
        //move right
}
```

Can you tell what should go in the statement part of the *else if* statement?

So we are now checking for input but we arent actually doing anything with it, in each of the separate blocks of code that the if statement creates we need to tell mario to move. The Player object has a function called *moveX*, this function allows us to move the player left and right. It takes one parameter which is how far the player should move. So to move left it is technically backwards along the x axis so the value should be minus whereas moving right is forwards so the value is positive. I suggest a movement of about 2 but feel free to play around with this.

So your if statement should now look something like this:

```
if(left.isDown()) {

        player.moveX(-2);

}
else if(right.isDown()) {

        player.moveX(2);

}
```

You have completed the basics of the course!