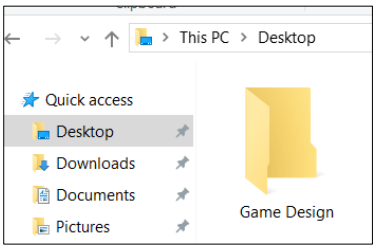
Game Design

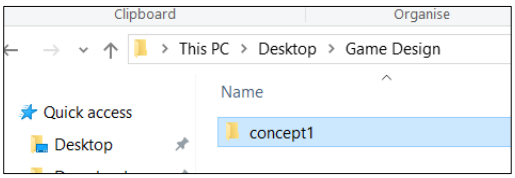
Concept 1 Tutorial

Build a website that loads a single image and demonstrates the game loop.

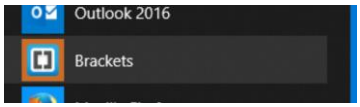
On your desktop create a new folder called Game Design



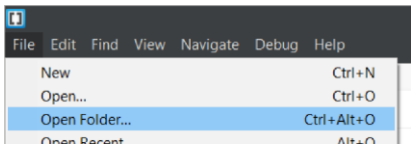
In this folder create a new folder called concept1



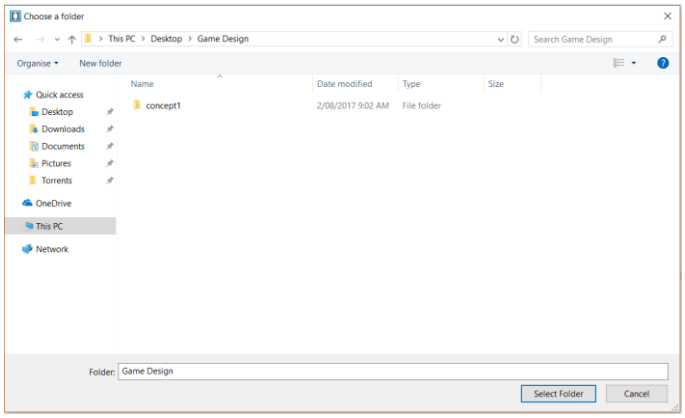
Open brackets



Click on open folder

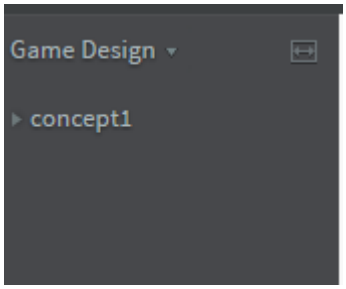


Select Games Design from the desktop

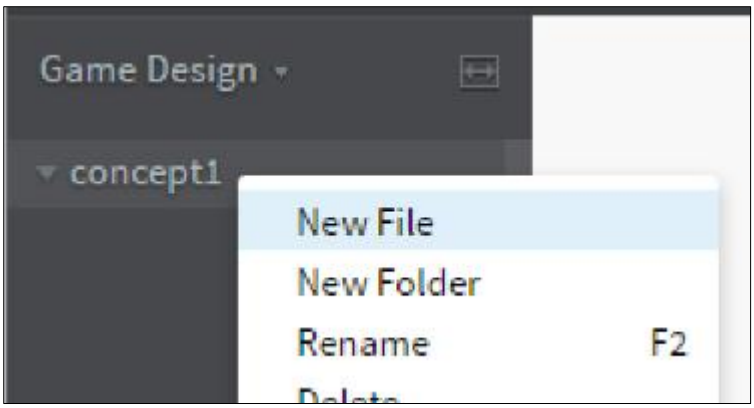


Once you see the above view, click on select folder

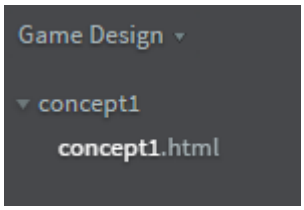
This will provide a menu system in brackets that looks like this



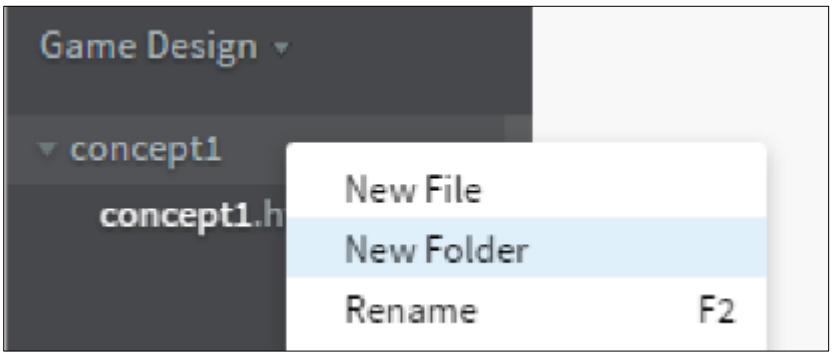
From here, click on the triangle next to concept to open up the folder, then right click inside concept1 and select new file



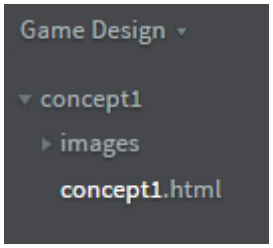
Name this file concept1.html



Create a new folder called images, right click on concept1 folder and select new folder

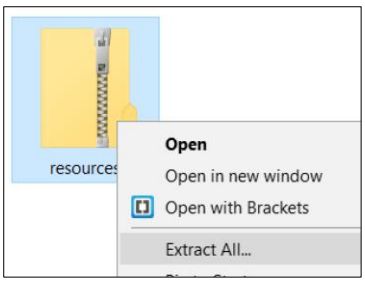


Name the folder images

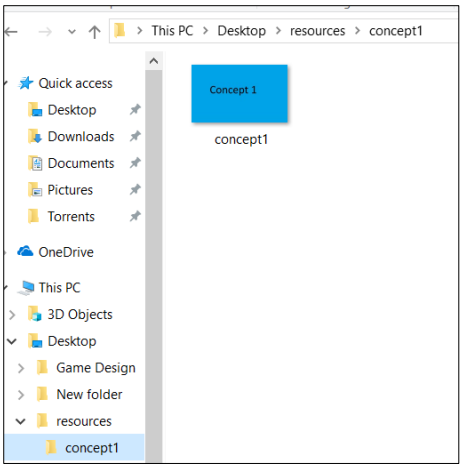


Download the resource file from http://www.angelshadowx.com/pd/resources.zip

Right click on the zip file and extract all items

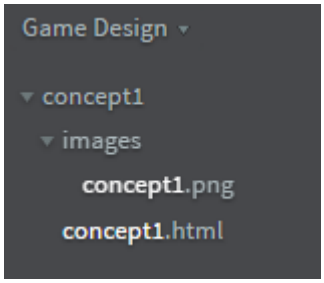


From there navigate to the newly created resource file, go into the concept 1 folder, you should see the following



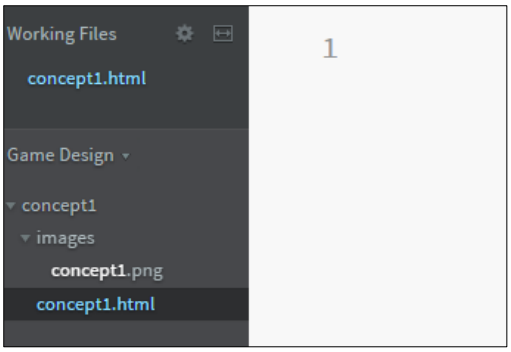
Next copy and paste the concept1.jpg into the games design/concept1/images folder

Now when we look inside brackets, if you expand the image folder you will see the following



Now we start to code up the concept of the game loop inside brackets.

Click on concept1.html, you will be presented with the following view



In the right-hand panel, type the following code

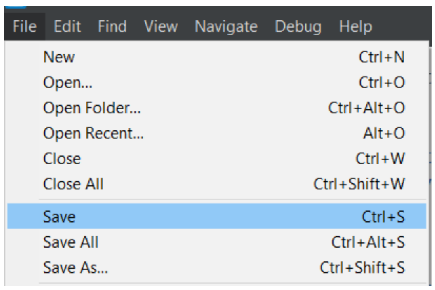
```
<!DOCTYPE html>
<html>
    <head>
        <title>Concept 1</title>
        <style type="text/css">
        </style>
    </head>
    <body>
        <script>
        </script>
    </body>
</html>
```

This is the base code that we will use for all the pages we create.

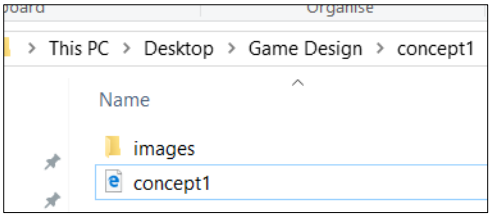
Now we will add some content and styles to the page.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Concept 1</title>
        <style type="text/css">
            .canvasColour{background-color: silver; margin-left: 50px; border-radius: 15px;}
            p{margin-left: 50px;}
            ul{margin-left: 50px;}
        </style>
    </head>
    <body>
        <canvas id="myCanvas" width="1000" height="600" class="canvasColour"></canvas>
        <p>To view Javascript code, each browser is unique in its opening of the developer tools.</p>
        <ul>
            <li>Edge: F12 or click on the 3 dots, then click on Developer tools</li>
            <li>Chrome: Ctrl+Shift+J or click on the 3 dots, then More tools, then Developer tools</li>
            <li>Firefox: Ctrl + Shift + K or click on the 3 bars, then Developer, then Web Console</li>
        </ul>
        <script>
        </script>
    </body>
</html>
```

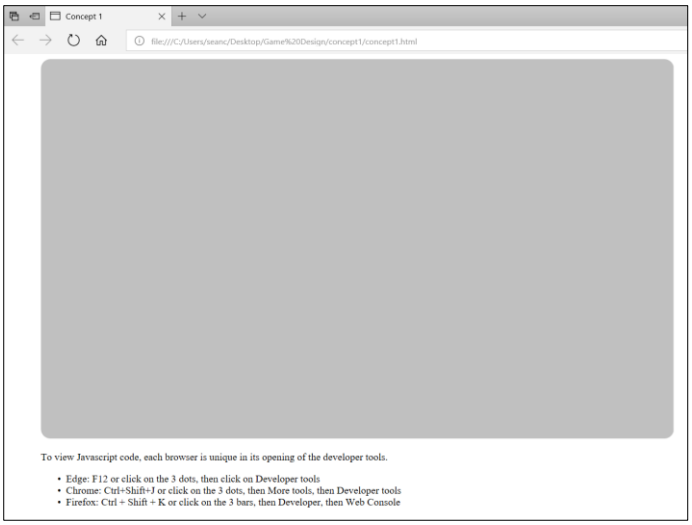
Once this is typed up, save the page



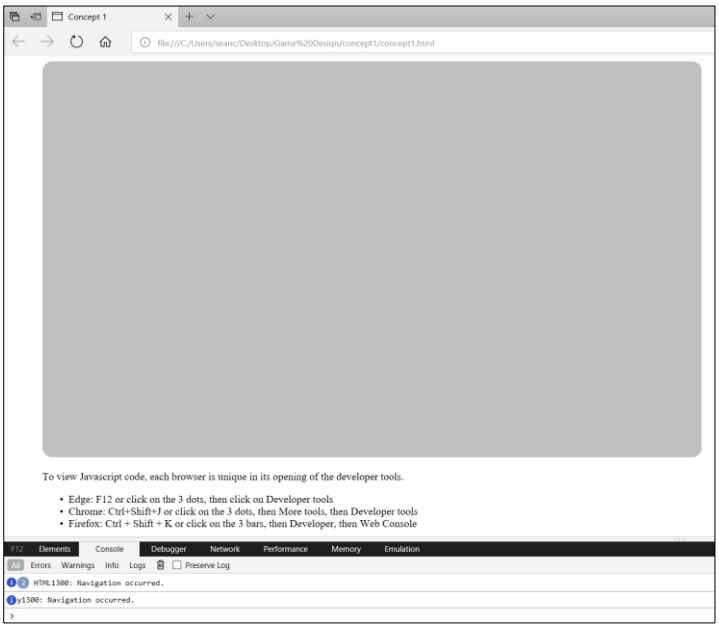
Next, go back to the desktop, go into the folder Game Design and then Concept1.



Double click on concept1.html. This will open the page in your default web browser(In my case Edge).



From here, open the console, you should see the following



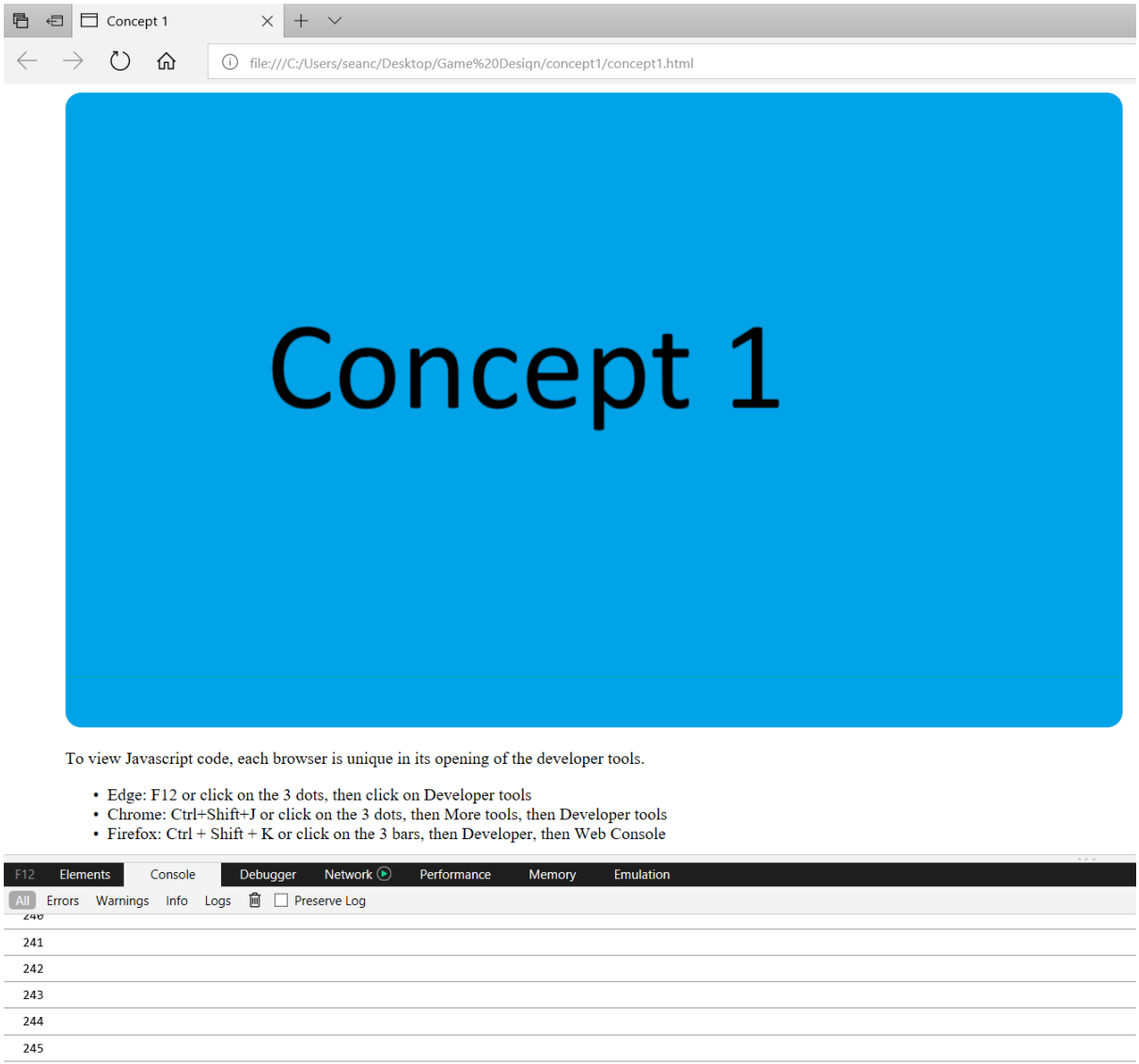
The console will be used to view the workings of JavaScript and allow issues to be located as well as tracking aspects of the code.

Next, back into brackets and we will start to add JavaScript to the page.

Code

```
        <script>
            var canvasObject = document.getElementById("myCanvas");
            var ctx = canvasObject.getContext('2d');
            var gameWidth = canvasObject.width;
            var gameHeight = canvasObject.height;
            //background image
            var bgImage = new Image();
            bgImage.src = "images/concept1.png";
            bgImage.addEventListener('load',init,false);
            var fps =120;

            window.requestAnimFrame = (function(){
                return  window.requestAnimationFrame       ||
                window.webkitRequestAnimationFrame ||
                window.mozRequestAnimationFrame    ||
                window.msRequestAnimationFrame     ||
                window.oRequestAnimationFrame      ||
                function( callback ){
                window.setTimeout(callback, 1000 / fps);
                };
            })();
                    var timer = 0;

            function init()
            {
                gameLoop();
            }

            function gameLoop()
            {
                setTimeout(function()
                {
                    requestAnimFrame(init);
                    clearScreen();
                    drawBackground();
                    timer++;
                    console.log(timer);
                },1000/fps);
            }

            function clearScreen()
            {
                ctx.clearRect(0,0,gameWidth,gameHeight);
            }
            function drawBackground()
            {
                ctx.drawImage(bgImage,0,0);
            }
        </script>
```

Save and test



When you run this up, you will notice that the console has a set of numbers that are continuously counting up, this is due to the function gameLoop() repeating all of the functions inside of it. This si the game loop we need to create engaging elements for a game.

Time to examine aspects of the code.

Code Breakdown

The below code allows the JavaScript to manipulate elements of the canvas.

```
var canvasObject = document.getElementById("myCanvas");
var ctx = canvasObject.getContext('2d');
var gameWidth = canvasObject.width;
var gameHeight = canvasObject.height;
```

The below code creates a variable that loads up the background image. The addEventListener line will instigate the init function once the background image has been loaded into memory.

```
//background image
var bgImage = new Image();
bgImage.src = "images/concept1.png";
bgImage.addEventListener('load',init,false);
```

The below code is the looping code that works across browsers. It's designed to eliminate the differences of animation looping that the different rendering engines have. The fps variable allows us some control over the frames per second in a browser.

```
var fps =120;

window.requestAnimFrame = (function(){
    return  window.requestAnimationFrame       ||
    window.webkitRequestAnimationFrame ||
    window.mozRequestAnimationFrame    ||
    window.msRequestAnimationFrame     ||
    window.oRequestAnimationFrame      ||
    function( callback ){
    window.setTimeout(callback, 1000 / fps);
    };
})();
```

The below variable exists so we can write something to the console

```
var timer = 0;
```

The below code is the init function, this is called once the background image has been loaded, and hence, start the game loop. It can be an area where you would set amount of lives, health levels and so forth before the game play starts.

```
function init()
{
    gameLoop();
}
```

The below code is the gameLoop, this is the primary function which Is redrawn a lot, it is where we manipulate graphics and determine user input. In this case, it calls the requestAnimFrame first, so this will work with out loop. Then a clearScreen function, drawBackground function, increment the timer variable by 1 and then write the timer information to the console. All whilst looping.

```
function gameLoop()
{
    setTimeout(function()
    {
        requestAnimFrame(init);
        clearScreen();
        drawBackground();
        timer++;
        console.log(timer);
    },1000/fps);
}
```

The below code clears the canvas of all graphical elements when run. This will remove ghost images during animation.

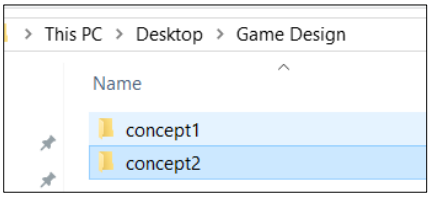
```
function clearScreen()
{
    ctx.clearRect(0,0,gameWidth,gameHeight);
}
```

The below code draws the background image. bgImage is the image object, where 0,0 is the X,Y co-ordinates of where to start drawing the item on the screen. 0,0 is the top left hand corner.

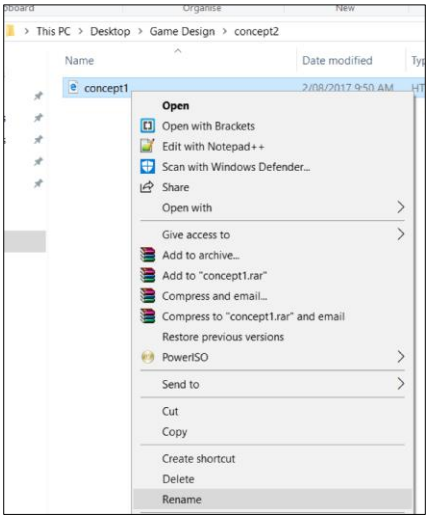
```
function drawBackground()
{
    ctx.drawImage(bgImage,0,0);
}
```

Concept 2 Tutorial

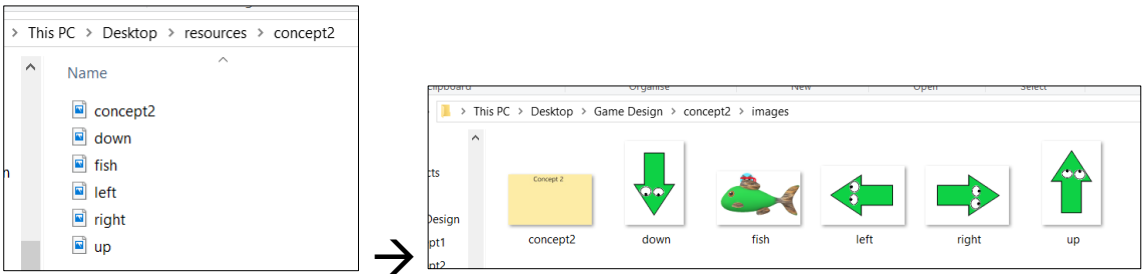
To start with, in the games design folder create a new folder called concept2



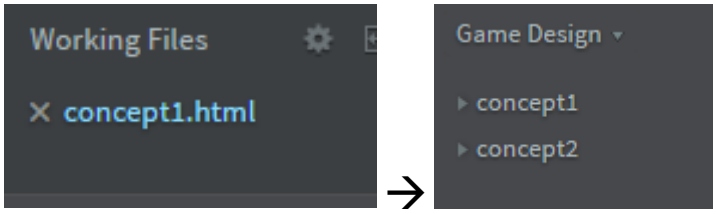
Now to speed up the coding, we can copy the concept1.html file into the concept2 folder and then rename it to concept2.html



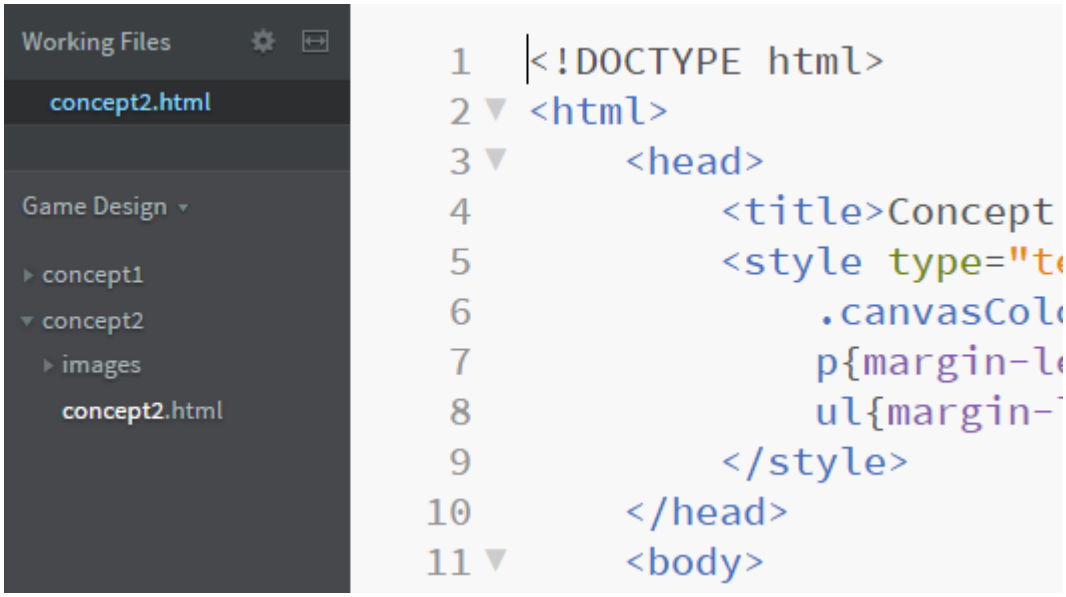
In addition, create the images folder and transfer the files from the resources\concept2 folder to the images folder.



Once this is done, return to brackets and close the concept1.html file (small x next to name) and minimise the concept1 folder.



Then expand the concept2 folder and double click concept2.html



From here we need to change a few little parts of the code before we start.

Let's change the title from concept 1 to concept 2

```
ead>
  <title>Concept 2</title>
  <style type="text/css">
```

To start with let's change the background image

```
//background image
var bgImage = new Image();
bgImage.src = "images/concept2.png";
bgImage.addEventListener('load',init,false);
var fps =120;
```

Then remove the timer variable and its console line.

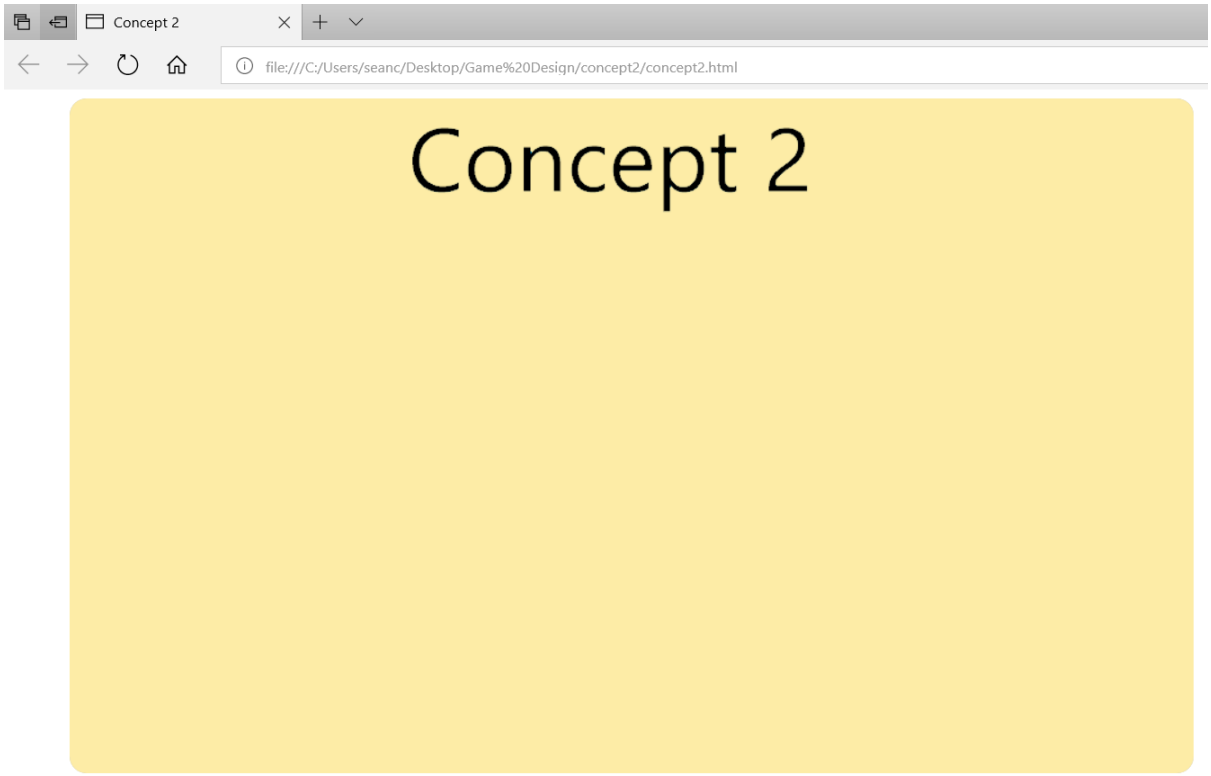
```
    window.setTimeout(callback, 1000 / fps);
    };
})();


function init()
```

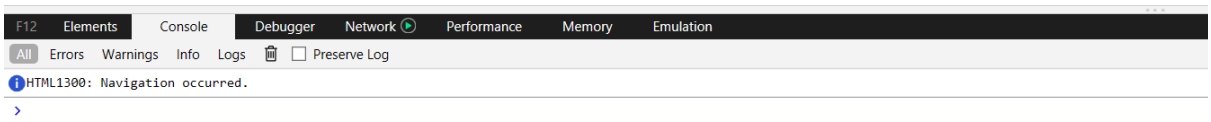
```
function gameLoop()
{
    setTimeout(function()
    {
        requestAnimFrame(init);
        clearScreen();
        drawBackground();
    },1000/fps);
}
```

We'll also remove the lines from the html telling us how to access the console.

```
<body>
    <canvas id="myCanvas" width="1000" height="600" class="canvasColour"></canvas>
    <p>To view Javascript code, each browser is unique in its opening of the developer tools.</p>
    <script>
```

If you save and test, you should see the following





Now we're going to add some more functions to our page. To start with, we will programmatically write some text to the canvas using a JavaScript function.

Add the following function

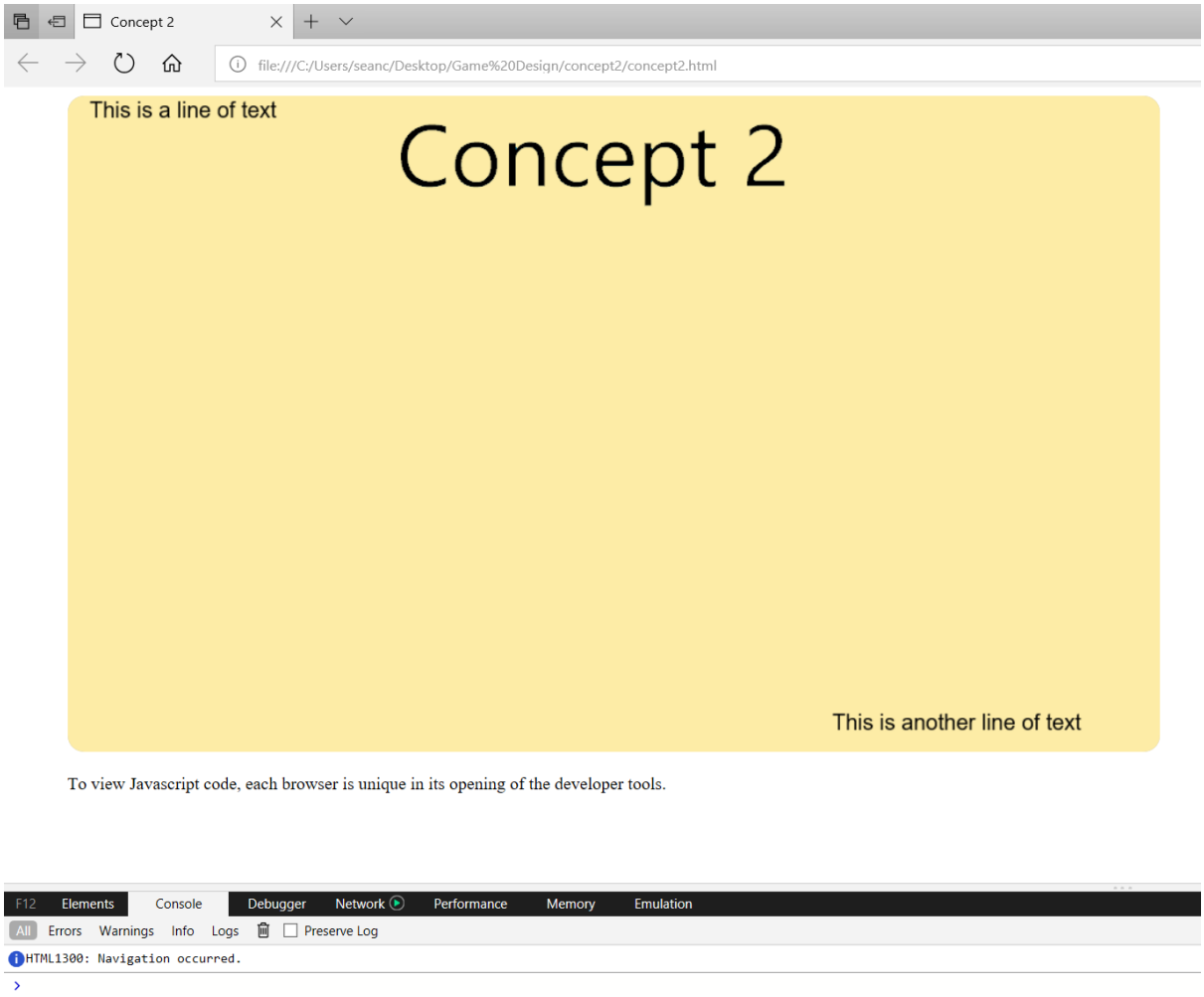
```
function drawText(text,x,y)
{
    ctx.fillStyle="black";
    ctx.font = "20px Arial";
    ctx.fillText(text,x,y);
}
```

This new function is designed to use parameters to pass information into it so it can then manipulate these elements. Even with this function now written and within the code, it is not going to do anything until it is called. Modify the gameLoop() function like so

```
function gameLoop()
{
    setTimeout(function()
    {
        requestAnimFrame(init);
        clearScreen();
        drawBackground();
        drawText("This is a line of text",20,20);
        drawText("This is another line of text",700,580);
    },1000/fps);
}
```

Once you have done this, make sure you save the page and then open concept2.html in a browser.

You should see the following



The x and y co-ordinates are feed into the function, so in the first line x is 20 pixels from the left, in the second line x is 700 pixels form the left. Y co-ordinate is the same, first line y is 20 pixels from the top and in the second line, y is 580 pixels from the top.

Next, we will some images onto the canvas. Use the following code

Global variables are used in this case, as such place them above the init function and below the loop code

```
//Arrows
var upA    = new Image();
upA.src    = "images/up.png";
var rightA = new Image();
rightA.src = "images/right.png";
var downA  = new Image();
downA.src  = "images/down.png";
var leftA  = new Image();
leftA.src  = "images/left.png";

//Fish
var fish = new Image();
fish.src = "images/fish.png";


function init()
```

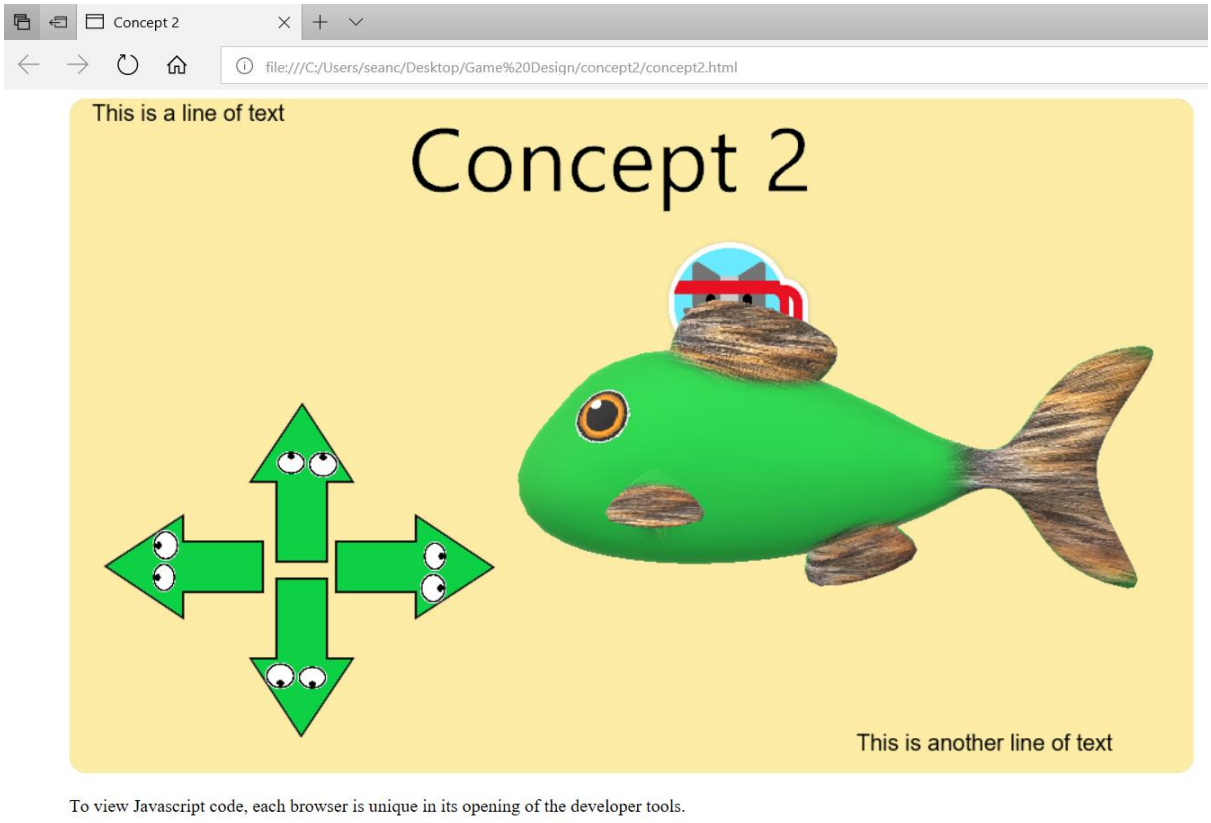
These variables load the images up into memory so we can use them. Next create the following two functions to draw the images onto the canvas.

```
function drawArrows()
{
    ctx.drawImage(upA,140,250);
    ctx.drawImage(rightA,210,350);
    ctx.drawImage(downA,140,400);
    ctx.drawImage(leftA,10,350);
}

function drawFish()
{
   ctx.drawImage(fish,370,120);
}
```

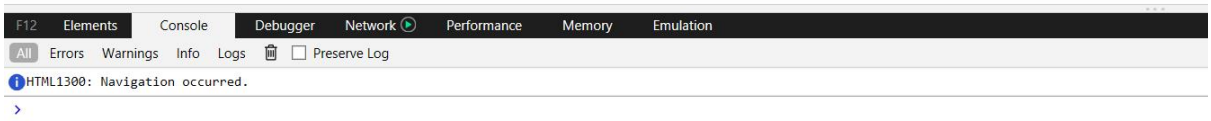
As before, even though the functions exist, we still need to use them, as such modify the gameLoop function like this

```
function gameLoop()
{
    setTimeout(function()
    {
        requestAnimFrame(init);
        clearScreen();
        drawBackground();
        drawText("This is a line of text",20,20);
        drawText("This is another line of text",700,580);
        drawArrows();
        drawFish();
    },1000/fps);
}
```

Once you have done this, save the page and load it in a browser, you should see the following



To view Javascript code, each browser is unique in its opening of the developer tools.



Now we are going to write some code that will allow us to move the fish by clicking on the arrows. To start with we need to capture any information that occurs when the mouse is clicked on the canvas. To do this we start with applying an onclick capability to the canvas.

```
<body>
    <canvas id="myCanvas" width="1000" height="600" class="canvasColour" onclick="handleClick(event)"></canvas>
```

From here, we write the handleClick function. Notice how we pass the event into this function, this is what gathers the event of clicking. To start with we will add a couple of global variables, a global variable is a variable that we can manipulate from any function.

```
//Fish
var fish = new Image();
fish.src = "images/fish.png";

//Mouse click location
var clickX;
var clickY;


function init()
```

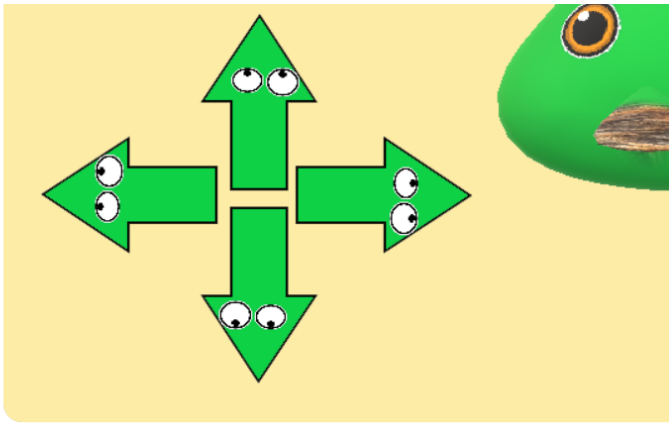
From here we then write the next function

```
function handleClick(event)
{
    var initX = event.clientX;
    var initY = event.clientY;
    var canvasX = canvasObject.offsetLeft - canvasObject.scrollLeft;
    var canvasY = canvasObject.offsetTop - canvasObject.scrollTop;
    clickX = initX - canvasX;
    clickY = initY - canvasY;
    console.log('x: ',clickX,' y: ',clickY);
}
```

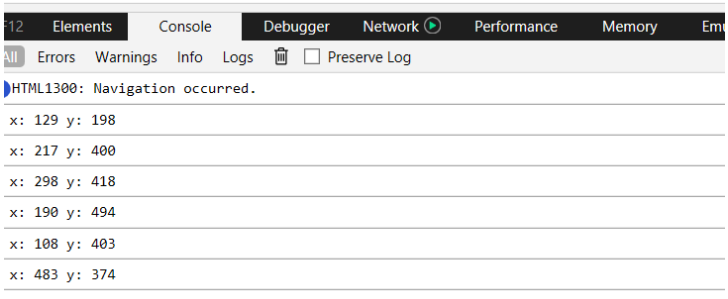
The handleClick function looks complicated but it is quite simple, it does the following:

- Collect the x,y co-ordinates of a mouse click
- Determine the offset (gap) between canvas and browser left edge
- Determine the offset (gap) between canvas and browser top
- Calculate the actual x,y values on the canvas removing the offset from the clicked point
- Writes these values to the console

When we save and go to the browser, and then click on the image, you will be able to see the x and y co-ordinates in the console. It should look like the following

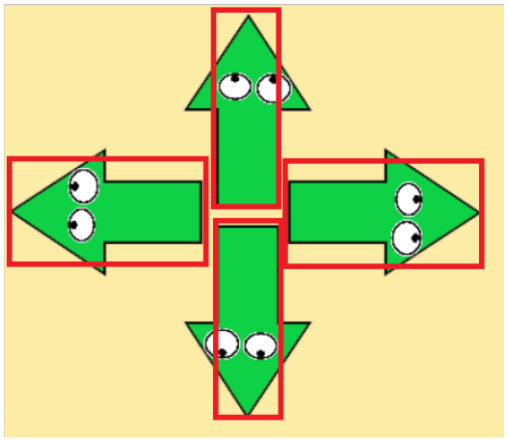


To view Javascript code, each browser is unique in its opening of the developer tool

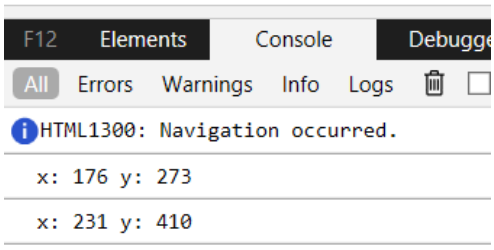


Next, we will have to determine where the clicks occur on the canvas, once we have that we can manipulate the fish's position.

But first we should figure out where we can click, to do this we gather the top left and bottom right x/y co-ordinates of the arrows. To do this go to the browser and click in the top left and bottom right of the up arrow.

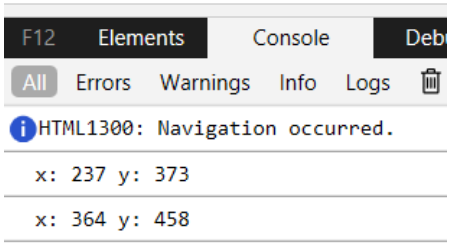


This should give you values such as:

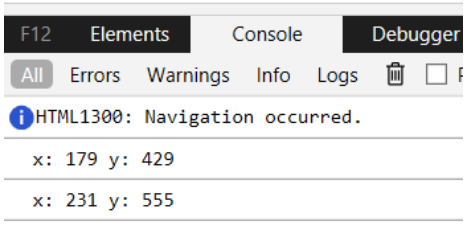


Up Arrow:

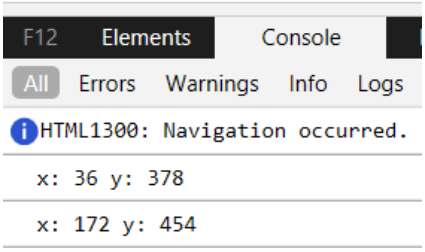
Repeat for the other arrows



Right Arrow:



Down Arrow:



Left Arrow:

With these elements located, we need to write a function to make the fish move. To start, we'll change the fish's x/y co-ordinates to global variables that can be changed.

```
//Fish
var fish = new Image();
fish.src = "images/fish.png";
var fishX = 370;
var fishY = 120;
```

These variables then need to be applied to the actual drawing of the fish. Like so

```
function drawFish()
{
    ctx.drawImage(fish,fishX,fishY);
}
```

Save and test, everything should still look the same.

From here, we create the moveFish function, this will be based off using if statements to control what we can do with the fish. To start with, let's write the code for moving the fish up.

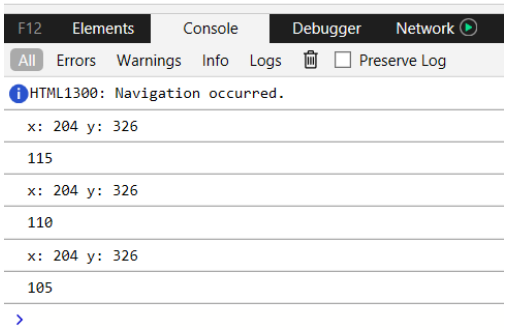
Write the following:

```
function moveFish()
{
    var fishSpeed = 5;

    if((clickX>=176) && (clickX<=231))
        {
            if((clickY>=273) && (clickY<=410))
                {
                    fishY = fishY - fishSpeed;
                    console.log(fishY);
                }
        }
}
```

Once this has been written, remember to add the moveFish() function to the handleClick function, like so

```
function handleClick(event)
{
    var initX = event.clientX;
    var initY = event.clientY;
    var canvasX = canvasObject.offsetLeft - canvasObject.scrollLeft;
    var canvasY = canvasObject.offsetTop - canvasObject.scrollTop;
    clickX = initX - canvasX;
    clickY = initY - canvasY;
    console.log('x: ',clickX,' y: ',clickY);
    moveFish();
}
```

Save and test, this should be able to move the fish, it's position should be numbered in the console to view.



Now that we have seen the fish move up, we add the remainder movement to the fish.

Modify moveFish() like so:

```
function moveFish()
{
    var fishSpeed = 5;
    //Up Arrow
    if((clickX>=176) && (clickX<=231))
    {
        if((clickY>=273) && (clickY<=410))
            {
                fishY = fishY - fishSpeed;
                console.log(fishY);
            }
    }
    //Right Arrow
    if((clickX>=237) && (clickX<=364))
    {
        if((clickY>=373) && (clickY<=458))
            {
                fishX = fishX + fishSpeed;
                console.log(fishY);
            }
    }
    //Down Arrow
    if((clickX>=179) && (clickX<=231))
    {
        if((clickY>=429) && (clickY<=555))
            {
                fishY = fishY + fishSpeed;
                console.log(fishY);
            }
    }
    //Left Arrow
    if((clickX>=36) && (clickX<=172))
    {
        if((clickY>=378) && (clickY<=458))
            {
                fishX = fishX - fishSpeed;
                console.log(fishY);
            }
    }
}
```

Once done, save and test. The fish should be able to be moved around via the arrows on the screen.

Last part of this concept is to add some animated bubbles for the ninja cat behind the fish. This can be done quite simply by adding moving circles to the canvas. But before we can do that, we need to be able to draw a circle.

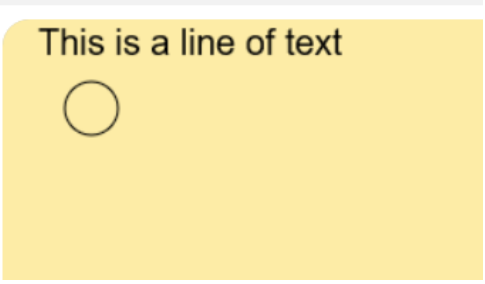
Type up the following function

```
function drawCircle(radius,x,y)
{
    ctx.beginPath();
    ctx.arc(x,y,radius,Math.PI*2,0,true);
    ctx.closePath();
    ctx.stroke();
}
```

And then test it in the game loop

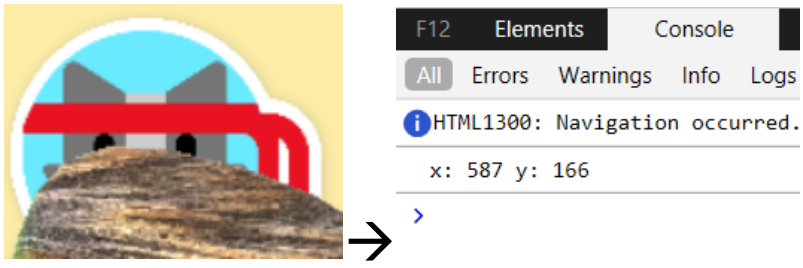
```
function gameLoop()
{
    setTimeout(function()
    {
        requestAnimFrame(init);
        clearScreen();
        drawBackground();
        drawText("This is a line of text",20,20);
        drawText("This is another line of text",700,580);
        drawArrows();
        drawFish();
        drawCircle(15,50,50);
    },1000/fps);
}
```

This should give you



In the top corner of the screen. So, now that we now that the circle function works, we will need to create bubble from the cat.

To start with, we'll calculate the position of the cat on the fish. This can be achieved by clicking in the centre of the cat. It gives you this information



What this tells us, is that the centre of the cat is approximately 200px further on than the start of the fish. If you recall, we gave the fish a starting x position of 370px, so I click 227px from the edge of the

fish. Now, the starting y position of the fish is 120, but a click on the bandana gave me 166, so a difference of 46px.

Now, create a new function called bubbles like this

```
function bubbles()
{
    var startBubblesX = fishX + randomBubble(200,240);
    var startBubblesY = fishY + randomBubble(20,50);
    var bubbleSize = randomBubble(5,20);
    drawCircle(bubbleSize,startBubblesX,startBubblesY);

}
function randomBubble(min,max)
{
    var randomNum = Math.floor((Math.random() * max) + min);
    return randomNum;
}
```

Notice the use of the secondary function, this is to introduce you to random functionality in JavaScript.

To make this work, we need to add back the timer variable and modify the gameLoop

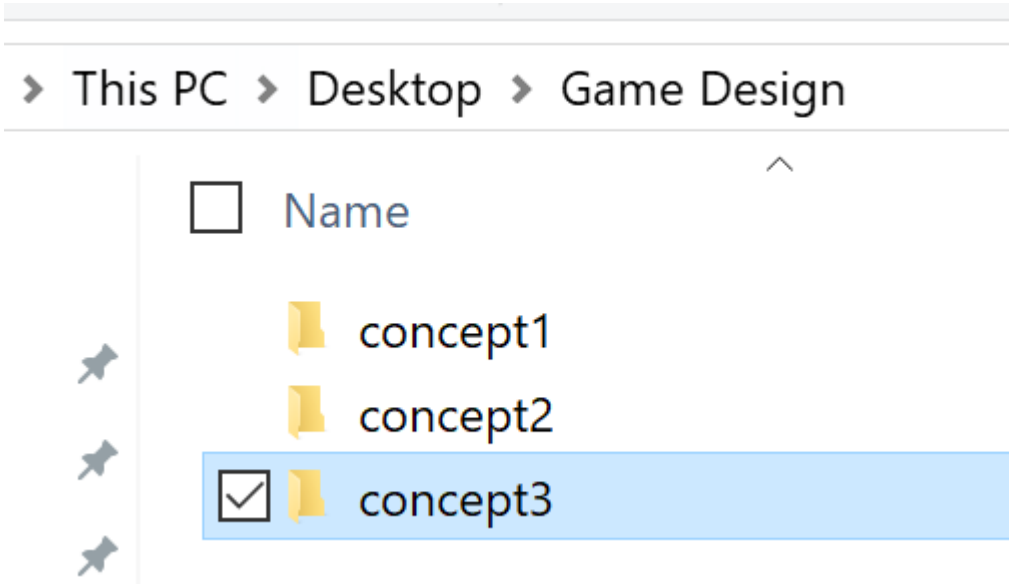
```
//bubbles
var timer=0;


function init()
{
```

```
function gameLoop()
{
    setTimeout(function()
    {
        requestAnimFrame(init);
        clearScreen();
        drawBackground();
        drawText("This is a line of text",20,20);
        drawText("This is another line of text",700,580);
        drawArrows();
        drawFish();
        timer++;
        if(timer%120 == 0)
            {
                setTimeout(bubbles(), 3000);
            }
    },1000/fps);
}
```

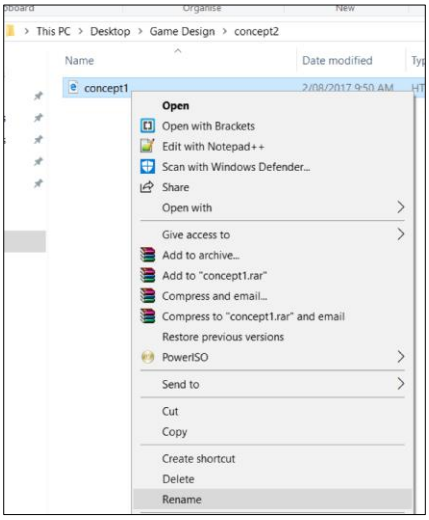
When this is saved and test, you will have little circles that pop up around the fish and cat, modifying the timer mod command and setTimeout will change the frequency of the bubbles. Each of the bubbles is only their briefly, to make them more consistent a differing method would be implemented.

Concept 3 Tutorial

To start with, in the games design folder create a new folder called concept2

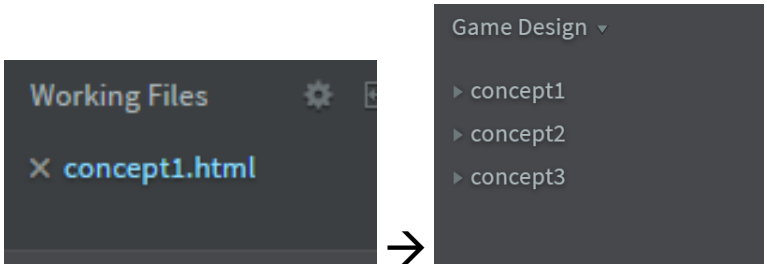


Now to speed up the coding, we can copy the concept1.html file into the concept3 folder and then rename it to concept3.html

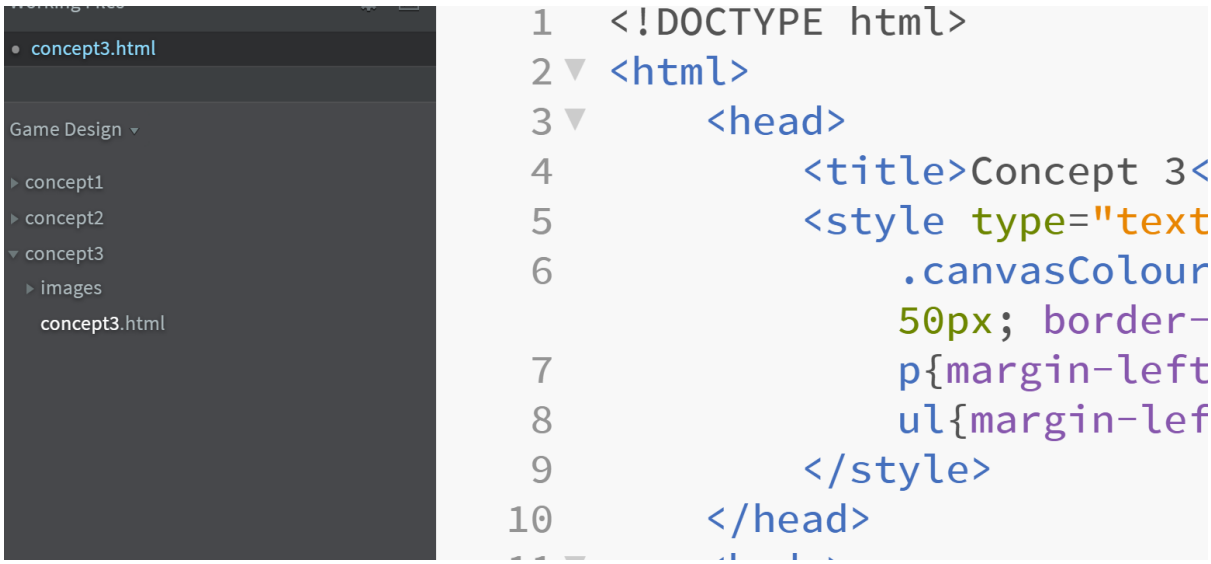


In addition, create the images folder and transfer the files from the resources\concept3 folder to the images folder.

Once this is done, return to brackets and close the concept1.html file (small x next to name) and minimise the concept1 folder.



Then expand the concept3 folder and double click concept3.html



From here we need to change a few little parts of the code before we start.

Let's change the title from concept 1 to concept 3

```
<title>Concept 3</title>
```

To start with let's change the background image

```
var bgImage = new Image();
bgImage.src = "images/concept3.png";
```

Then remove the timer variable and its console line.

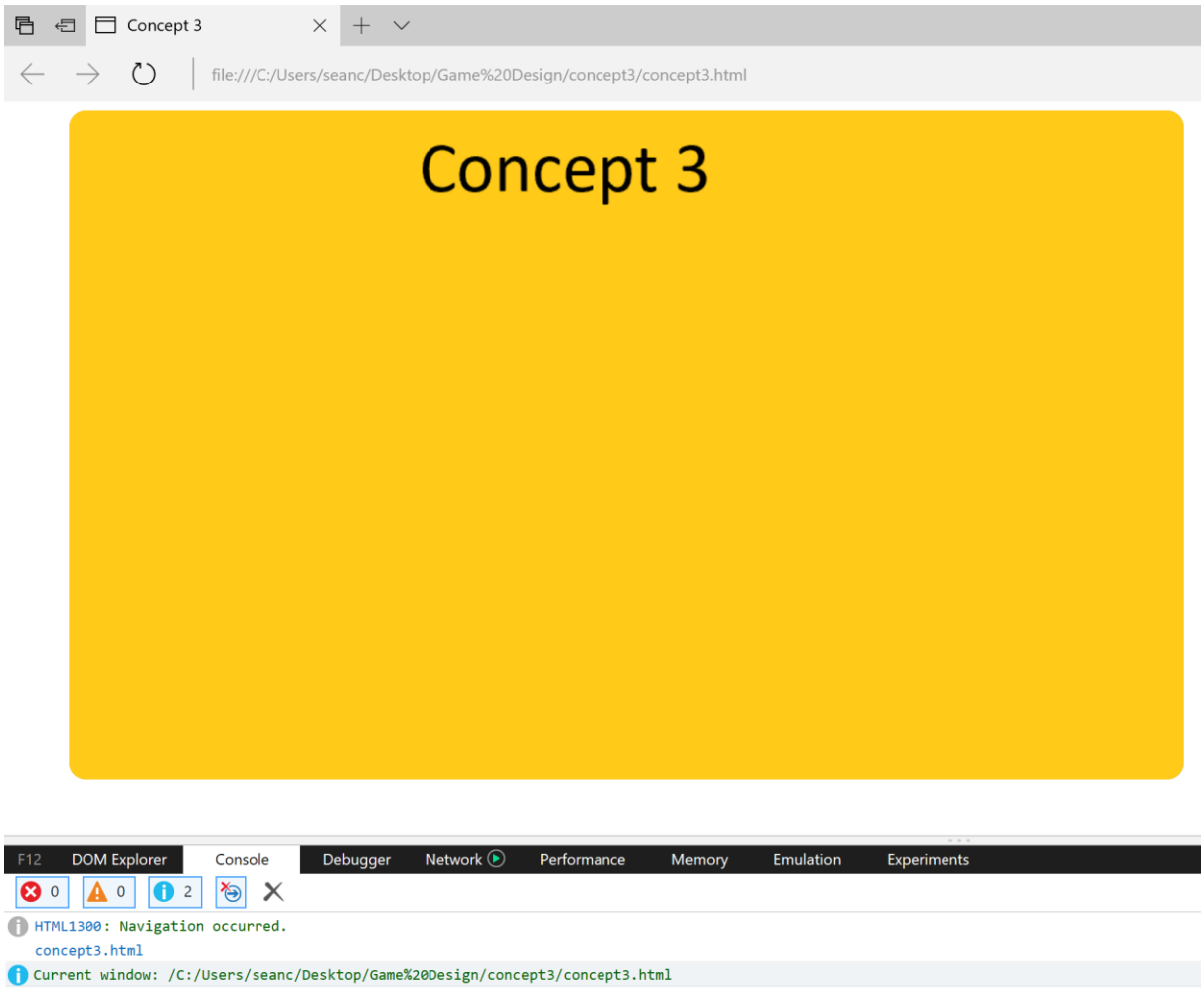
```
    window.setTimeout(callback, 1000 / fps);
    };
})();


function init()
```

```
function gameLoop()
{
    setTimeout(function()
    {
        requestAnimFrame(init);
        clearScreen();
        drawBackground();
    },1000/fps);
}
```

We'll also remove the lines from the html telling us how to access the console.

```
<body>
    <canvas id="myCanvas" width="1000" height="600"
    class="canvasColour"></canvas>
    <script>
```

If you save and test, you should see the following



Now that we have this structure in place, it's time to start loading the environment that is needed to complete this concept.

To start with, there will be the loading of the images.

Use the following code:

The below code contains global variables to enable manipulation from any function, later on it the program.

```
//player (Praying Mantis)
var playerX = 20;
var playerY = 510;
var pMantis = new Image();
pMantis.src = "images/prayingMantis.png";

//NPC 1 (fly)
var flyX = 10;
var flyY = 10;
var fly = new Image();
fly.src = "images/fly.png";

//NPC 2 (spider)
var spiderX = 800;
var spiderY = 500;
var spider = new Image();
spider.src = "images/spider.png";
```

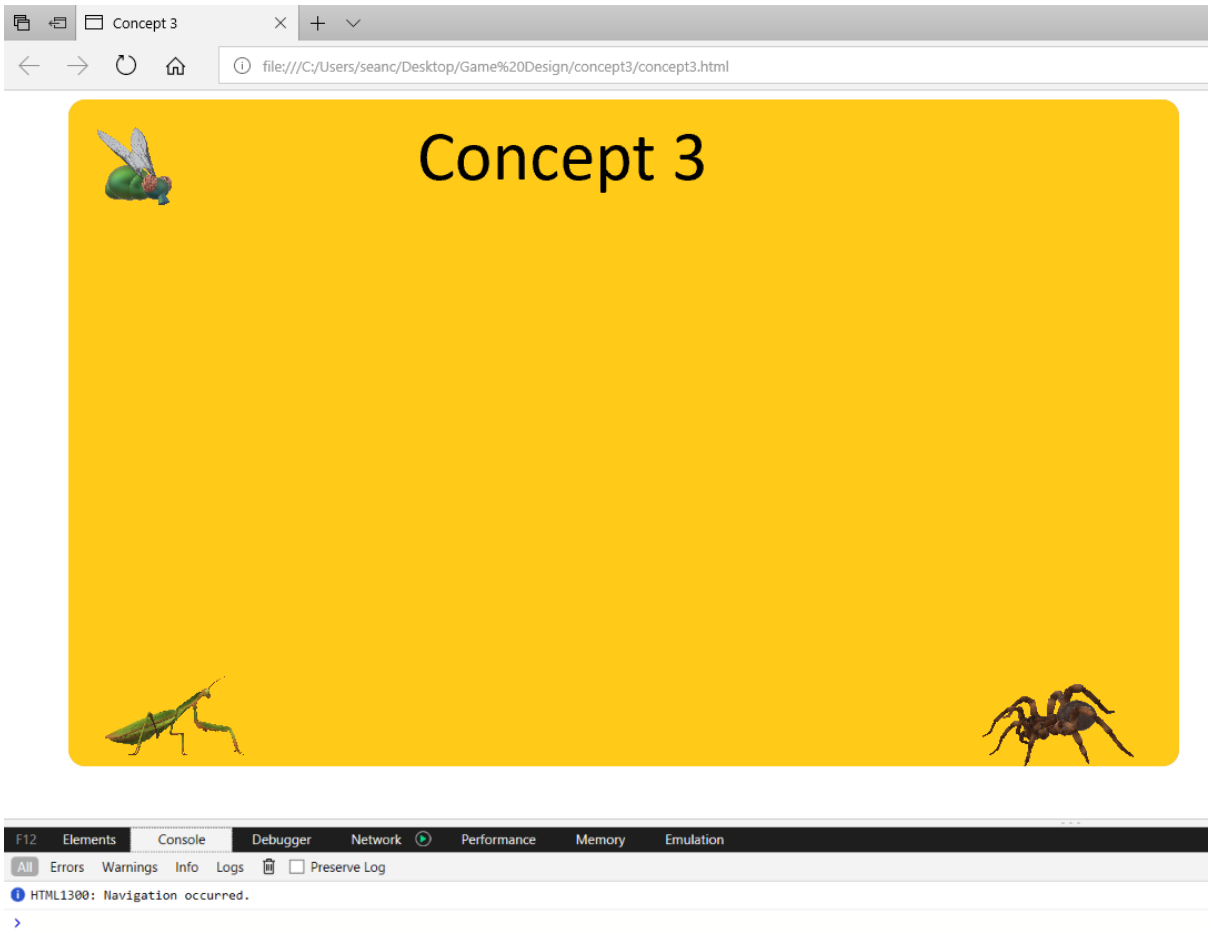
The below code is the functions that actually draw the elements on to the canvas.

```
function drawPrayingMantis()
{
    ctx.drawImage(pMantis,playerX,playerY);
}

function drawFly()
{
    ctx.drawImage(fly,flyX,flyY);
}

function drawSpider()
{
    ctx.drawImage(spider,spiderX,spiderY);
}
```

The next change is a modification to the gameloop code to ensure that the functions runs

```
function gameLoop()
{
    setTimeout(function()
    {
        requestAnimFrame(init);
        clearScreen();
        drawBackground();
        drawPrayingMantis();
        drawFly();
        drawSpider();
    },1000/fps);
}
```

Save and test, you should see the following



Now we are going to introduce the mouse over capability, where the praying mantis will follow the mouse around the screen. To do this, we need to capture the x/y co-ordinates and feed them into the praying mantis's position.

To start with, change the code on the canvas

```
<canvas id="myCanvas" width="1000" height="600" class="canvasColour" onmousemove="captureMouse(event)"></canvas>
```

Now, add the following function. The below code captures the x/y of the mouse, then it applies a position change of the cursor to allow for the cursor to centre on the praying mantis image.

```
function captureMouse(event)
{
    var x = event.clientX;
    var y = event.clientY;
    spriteX = pMantis.width;
    spriteY = pMantis.height/2;
    playerX = x - spriteX;
    playerY = y - spriteY;
}
```

Save and test, this creates a praying mantis that follows the cursor on the screen.

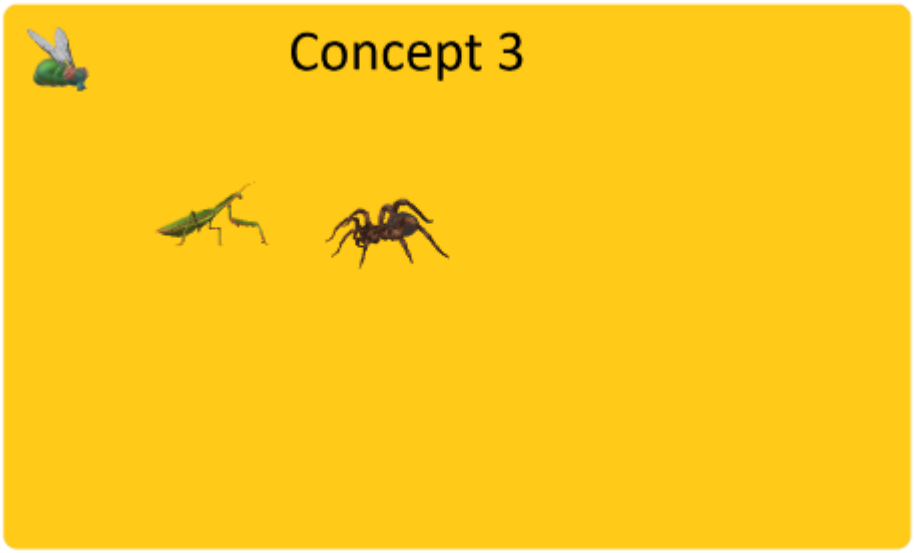
Next, we will make the spider follow the praying mantis, to do this we will add the following code

```
function moveSpider()
{
    var spiderSpeed = 3;
    if(spiderX>= playerX){
        spiderX = spiderX - spiderSpeed;
    }else if(spiderX<= playerX){
        spiderX = spiderX + spiderSpeed;
    }
    if(spiderY>= playerY){
        spiderY = spiderY - spiderSpeed;
    }else if(spiderY<= playerY){
        spiderY = spiderY + spiderSpeed;
    }
}
```

Even with the function created, we still need to activate it, to do so, we make a change to gameLoop

```
function gameLoop()
{
    setTimeout(function()
    {
        requestAnimFrame(init);
        clearScreen();
        drawBackground();
        drawPrayingMantis();
        drawFly();
        drawSpider();
        moveSpider();
    },1000/fps);
}
```

Now, save an test, you should have the spider chasing down the praying mantis, it will look a little like this



What we want to do now, is to know when the spider captures the mantis, as such, we need to introduce collision detection. When we view the images they look perfectly normal, but they are surrounded by a box, if you can visualise this, it would like this this:

Our next step is to determine when these boxes run into each other. To do this, we take the x/y co-ordinate of each image, then calculate the box by using the images height and width. Once we have these positions, we will introduce a Boolean variable to the code that will indicate when the collision occurs.

This is done by the following code. We add a global variable so we can manipulate aspects of the game based off collision.

```
//collision
var hit = false;
```

Then we add the following collision code.

```
function msCollision()
{
    var pMantisWidth = (playerX+pMantis.width) - 80;
    var pMantisHeight = playerY+pMantis.height;
    var spiderWidth = spiderX+spider.width;
    var spiderHeight = spiderY+spider.height;
    if((spiderX<=pMantisWidth) && (spiderX>= playerX)
       && (spiderY>=playerY) && (spiderY<=pMantisHeight))
        {
            hit = true;
        }
}
```

Notice the -80 added to the praying mantis's width, this is to let the spider end up more on the mantis, otherwise there would be a gap between the images.

And modify the gameLoop code like the following

```
function gameLoop()
{
    if(!hit)
    {
        setTimeout(function()
        {
            requestAnimFrame(init);
            clearScreen();
            drawBackground();
            drawPrayingMantis();
            drawFly();
            drawSpider();
            moveSpider();
            msCollision();
        },1000/fps);
    }
}
```

So, what we have achieved here is the gameLoop stopping when the spider catches the praying mantis. So Boolean variables have two states, true or false. And when coding, we have the advantage of writing if(Boolean) to indicate true instead of if (Boolean == true), the exclamation mark(!) means not, so if(!hit) translates to if not true; with not true being false.

Therefore, we set the global variable as false so we would be able to enter the game loop to start with, once the collision has occurred, we change the Boolean value from false to true.

Now we will add a score capability and start moving the fly around.

For a score we will grab the drawText function from concept 2 and copy it in, and create a global variable.

```
//score
var score = 0;
```

The drawText function, with a slightly lowered font size

```
function drawText(text,x,y)
{
    ctx.fillStyle="black";
    ctx.font = "18px Arial";
    ctx.fillText(text,x,y);
}
```

And we will add an updateScore function to write this information to the screen. It looks like this:

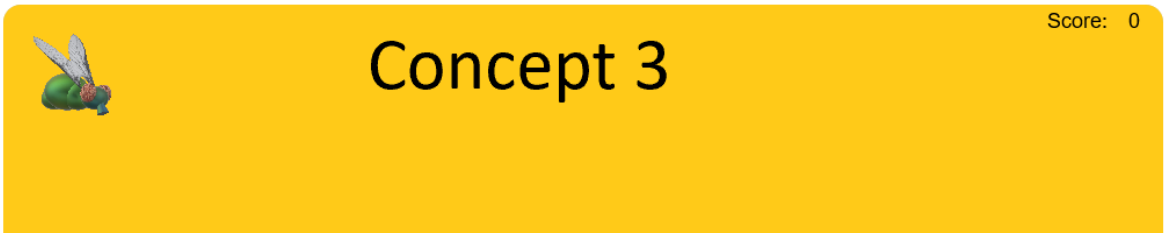
```
function updateScore()
{
    drawText("Score: ",900,20);
    drawText(score,970,20);
}
```

So, once again we have elements in the code that are ready to be used, but we still need to add them to the gameLoop so we can see what is going on with it.

Modify gameLoop like this

```
function gameLoop()
{
    if(!hit)
    {
        setTimeout(function()
        {
            requestAnimFrame(init);
            clearScreen();
            drawBackground();
            drawPrayingMantis();
            drawFly();
            drawSpider();
            moveSpider();
            msCollision();
            updateScore();
        },1000/fps);
    }
}
```

Save and test, you should see the following score appear in the top right-hand corner:



Next, we will move the fly around the screen. To make this different from the spider, we will get it to randomly run around but box it into the canvas. To do this, we use the following code:

```
function moveFly()
{
    var flySpeed = 20;
    var flyDirection = Math.round(((Math.random()* 4) +1));
    //console.log(flyDirection);
    switch(flyDirection){
        case 1: flyY = flyY-flySpeed; break;
        case 2: flyX = flyX+flySpeed; break;
        case 3: flyY = flyY+flySpeed; break;
        case 4: flyX = flyX-flySpeed; break;
    }
    if(flyX<=10){flyX = 10;}
    if(flyX>=950){flyX = 950;}
    if(fly<=10){flyY = 10;}
    if(flyY>=550){flyY = 550;}

}
```

This introduces the switch code, where we can examine a variable and then determine an action, it's a cleaner way of writing multiple if statements. We also need to turn it on in the gameLoop, like so:

```
function gameLoop()
{
    if(!hit)
    {
        setTimeout(function()
        {
            requestAnimFrame(init);
            clearScreen();
            drawBackground();
            drawPrayingMantis();
            drawFly();
            drawSpider();
            moveSpider();
            msCollision();
            moveFly();
            updateScore();
        },1000/fps);
    }
}
```

When you run this, the fly will jitter around to changing his position every loop, which is a lot like a fly. Next, we want to have to praying mantis catch the fly and increase our score. To do this, we will need to implement a collision for the praying mantis and fly. This can be done using the following code:

Start with a new global variable

```
//collision
var hit = false;
var eatFly = false;
```

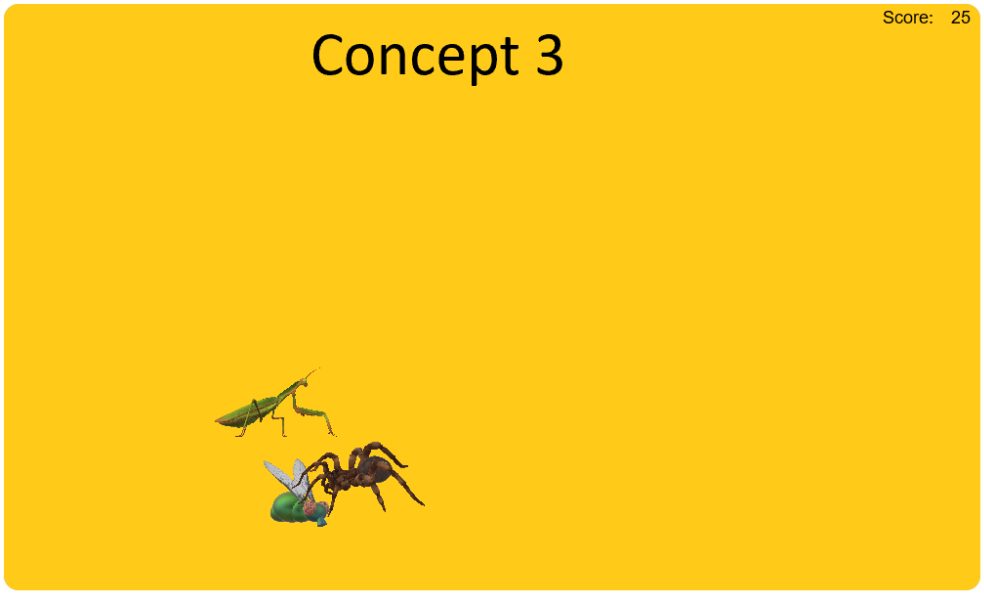
Add the fly collision code

```
function flyCollision()
{
    var pMantisWidth = (playerX+pMantis.width) - 80;
    var pMantisHeight = playerY+pMantis.height;
    var flyWidth = flyX+fly.width;
    var flyHeight = flyY+fly.height;
    if((flyX<=pMantisWidth) && (flyX>= playerX)
       && (flyY>=playerY) && (flyY<=pMantisHeight))
        {
            eatFly = true;
            score++;
        }
}
```

Then modify the game loop code

```
function gameLoop()
{
    if(!hit)
    {
        setTimeout(function()
        {
            requestAnimFrame(init);
            clearScreen();
            drawBackground();
            drawPrayingMantis();
            drawFly();
            drawSpider();
            moveSpider();
            msCollision();
            moveFly();
            flyCollision();
            updateScore();
        },1000/fps);
    }
}
```

Save and test, when you run the praying mantis over the fly your score goes up until the spider catches the praying mantis.



Which is good, except that by keeping the praying mantis over the fly just increases the score, so now we make some modifications to re-position the fly after the collision has taken place. To make it more interesting, we will randomise the position.

Create the following code

```
function bounceFly()
{
    flyX = ((Math.random()*900) + 10);
    flyY = ((Math.random()*500)+20);
    eatFly = false;
}
```

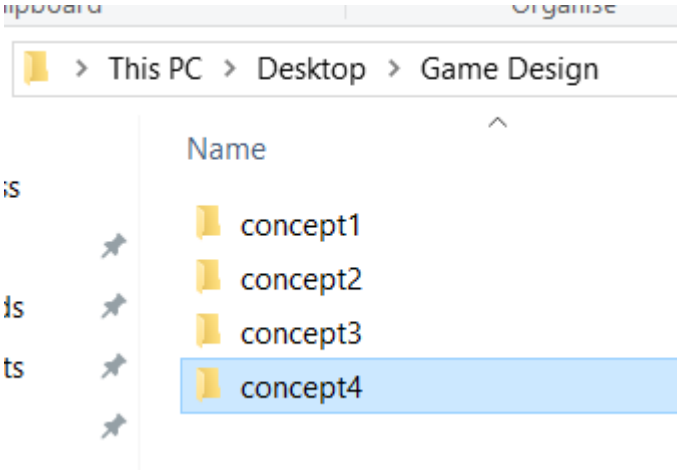
Modify the gameLoop

```
function gameLoop()
{
    if(!hit)
    {
        setTimeout(function()
        {
            requestAnimFrame(init);
            clearScreen();
            drawBackground();
            drawPrayingMantis();
            drawFly();
            drawSpider();
            moveSpider();
            msCollision();
            if(eatFly)
                bounceFly();
            moveFly();
            flyCollision();
            updateScore();
        },1000/fps);
    }
}
```

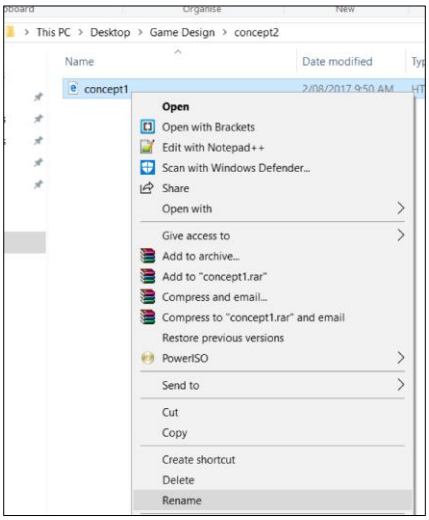
Save and test. This should give you a fly that bounces around the canvas after being caught.

Concept 4 Tutorial

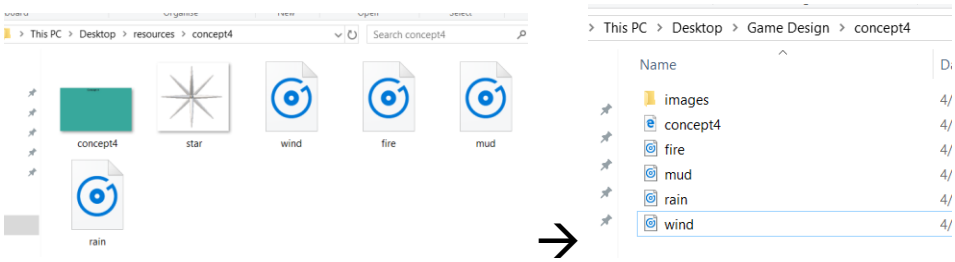
To start with, in the games design folder create a new folder called concept4



Now to speed up the coding, we can copy the concept1.html file into the concept4 folder and then rename it to concept4.html

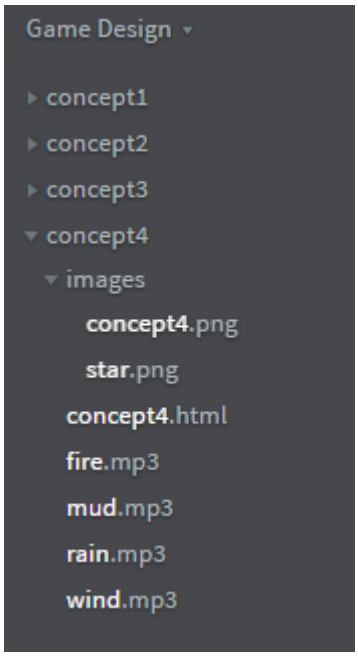


In addition, create the images folder and transfer the files from the resources\concept4 folder to the images folder.



Once this is done, move the images into the images folder and leave the audio files next to concept4.html. Return to brackets.

Then expand the concept4 folder and it's images folder, you should have the following.



From here we need to change a few little parts of the code before we start.

Let's change the title from concept 1 to concept 4

```
<title>Concept 4</title>
```

To start with let's change the background image

```
//background image
var bgImage = new Image();
bgImage.src = "images/concept4.png";
bgImage.addEventListener('load',init,false);
```

Then remove the timer variable and its console line.

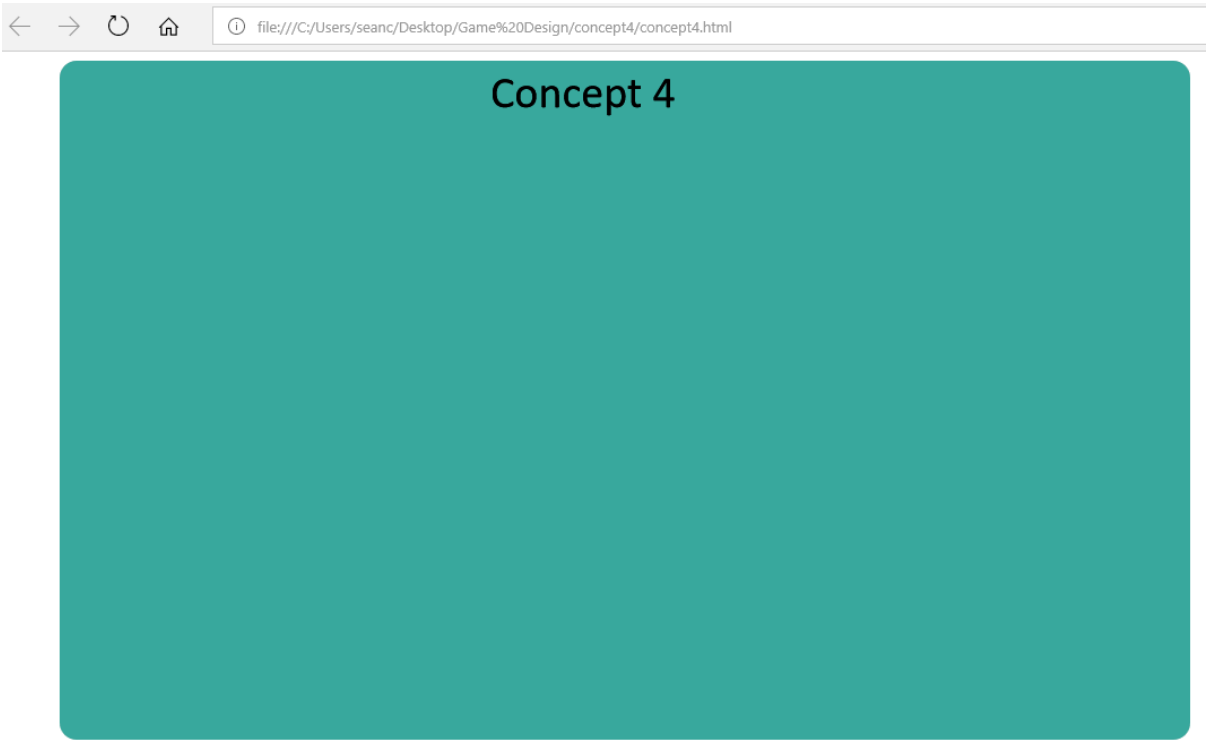
```
    window.setTimeout(callback, 1000 / fps);
    };
})();


function init()
```

```
function gameLoop()
{
    setTimeout(function()
    {
        requestAnimFrame(init);
        clearScreen();
        drawBackground();
    },1000/fps);
}
```

We'll also remove the lines from the html telling us how to access the console.

```
<body>
    <canvas id="myCanvas" width="1000" height="600"
    class="canvasColour"></canvas>
    <script>
```

If you save and test, you should see the following



From here we are going to add our star to the canvas, though instead of being a solitary star on the canvas. It will be implemented within an array, so we can have multiple versions of the same element. To do this, we need to create some global variables.

The primary difference in how we are doing this is that we will be creating variables that allow us to modify the size of the image on the canvas.

Global Variables

```
//Star
var star = new Image();
star.src = "images/star.png";
starX = 50;
starY = 50;
starSize = 150;
```

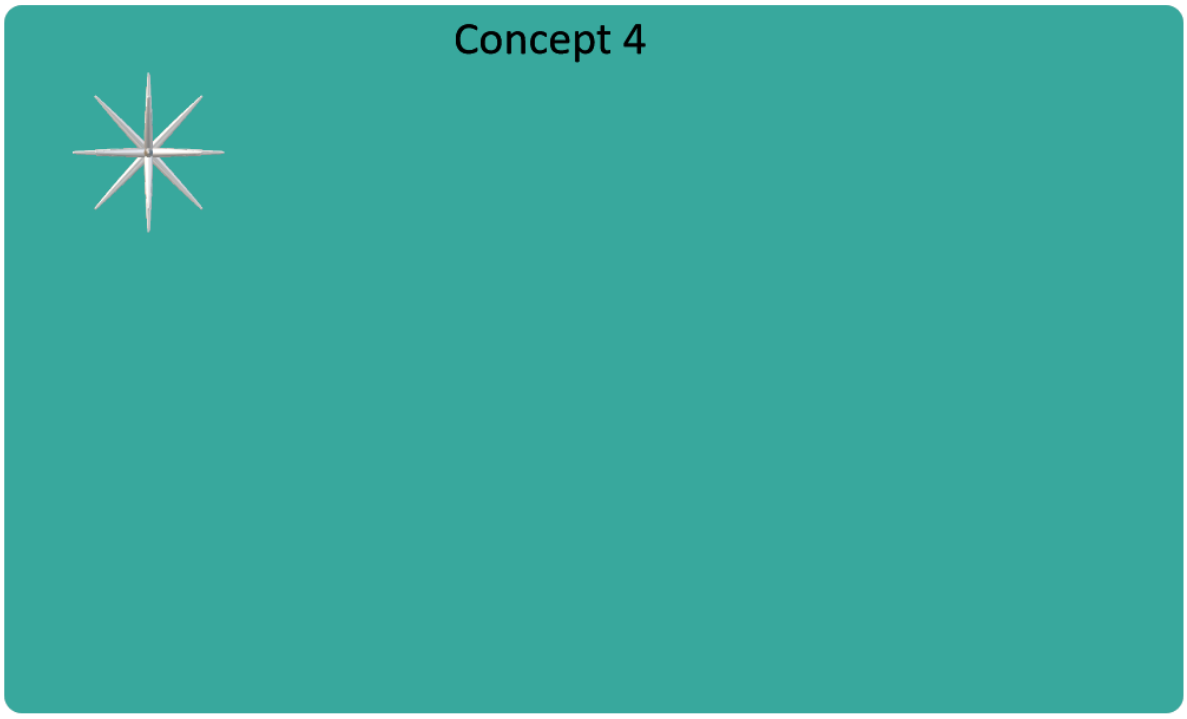
Function to draw the star

```
function drawStar()
{
     ctx.drawImage(star,starX,starY,starSize,starSize);
}
```

Gameloop modification

```
function gameLoop()
{
    setTimeout(function()
    {
        requestAnimFrame(init);
        clearScreen();
        drawBackground();
        drawStar();
    },1000/fps);
}
```

If we save and test this, you should see the following



From here, we are going to implement 4 stars, to do this we will be using an array to assign location, to start with.

Write the following code:

Modify the global variables

```
//Star
var star = new Image();
star.src = "images/star.png";
var starX = [50,250,500,700];
var starY = [50,150,250,350];
var starSize = 150;
```

Then the drawStar function

```
function drawStar()
{
    for(item=0;item<=4;item++)
    {
        ctx.drawImage(star,starX[item],starY[item],starSize,starSize);
    }
}
```

As you can see, this is using a for loop, so each time the loop iterates it moves to a new number in the array. It works like this table:

Iteration	starX value	starY value
0	50	50
1	250	150
2	500	250
3	700	350

This progression allowed the program to store multiple values under the one name.

To make this more interesting, we'll add some randomness to the program, so the stars will be in various locations and assorted sizes.

Modify the code as below.

Change the global variables to:

```
//Star
var star = new Image();
star.src = "images/star.png";
var starX = [];
var starY = [];
var starSize = [];
var maxStars = 4;
```

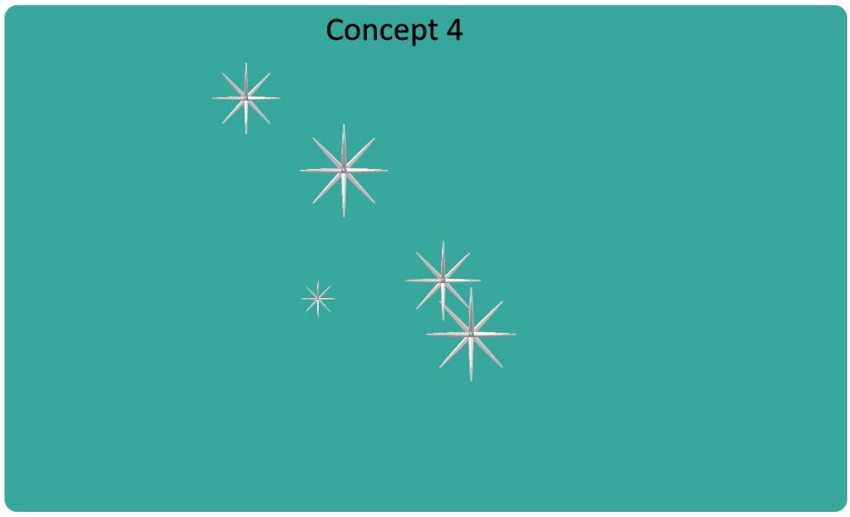
Then modify the drawstars function like this

```
function drawStar()
{
    for(item=0;item<=maxStars;item++)
    {
        starX[item] = randomNumber(20,800);
        starY[item] = randomNumber(20,400);
        starSize[item] = randomNumber(25,100);
        ctx.drawImage(star,starX[item],starY[item],starSize[item],starSize[item]);
    }
}
```

And finally, add the new random number function

```
function randomNumber(min,max)
{
    r = Math.round((Math.random()*max)+min);
    return r;
}
```

If you save and test this will showcase a screen with a lot of moving stars. Such a:



If you wanted to you could slow down the flashing stars by dropping the global variable fps to a smaller number. As fun as this was, we need to only have a few stars, so we will pull the random calls out of the draw star function into a new function and place it under the global variables. In this way the star parameters are calculated only once.

Make the following changes:

```
//Star
var star = new Image();
star.src = "images/star.png";
var starX = [];
var starY = [];
var starSize = [];
var maxStars = 3;
starInitilise();
```

Note the changing of maxStars to 3 from 4. This is because we will add audio to each star.

The drawstar function is cleaned up to:

```
function drawStar()
{
    for(item=0;item<=maxStars;item++)
    {
        ctx.drawImage(star,starX[item],starY[item],starSize[item],starSize[item]);
    }
}
```

And the new function created looks like this:

```
function starInitilise()
{
    for(item=0;item<=maxStars;item++)
    {
        starX[item] = randomNumber(20,800);
        starY[item] = randomNumber(20,400);
        starSize[item] = randomNumber(25,100);
    }
}
```

These changes produce a static set of 4 stars each time the page is refreshed.

Now we will add the audio tracks to the page. To start with, we have to link the audio to the page and to achieve this we will use a 3rd party addition from a company called createjs. By using this plugin, it is possible to play multiple audio streams at once, which will be useful.

To link our audio tracks add the following.

```
<canvas id="myCanvas" width="1000" height="600" class="canvasColour"></canvas>
<script src="https://code.createjs.com/createjs-2015.11.26.min.js"></script>
<script>
```

Without that line of code, we won't be able to access the createjs JavaScript files.

Now we start adding in global variables to let us access the mp3 files

```
//Audio
var soundID_Fire = "fire";
var soundID_Mud = "mud";
var soundID_Rain = "rain";
var soundID_Wind = "wind";
loadSound();
```

The loadSound that is added to the global variable area, so to speak will activate the following function, that we must add:

```
function loadSound()
{
    createjs.Sound.registerSound("fire.mp3", soundID_Fire);
    createjs.Sound.registerSound("mud.mp3",soundID_Mud);
    createjs.Sound.registerSound("rain.mp3",soundID_Rain);
    createjs.Sound.registerSound("wind.mp3",soundID_Wind);
}
```

Now we should capture the mouse click, so we have to add a modification to the canvas like this:

```
<canvas id="myCanvas" width="1000" height="600" class="canvasColour" onclick="handleClick(event)"></canvas>
```

From here, create the handleClick function

```
function handleClick(event)
{
    var posX = event.clientX;
    var posY = event.clientY;
    var offSetX = canvasObject.offsetLeft - canvasObject.scrollLeft;
    var offSetY = canvasObject.offsetTop - canvasObject.scrollTop;
    posX = posX - offSetX;
    posy = posY - offSetY;
    activateAudio(posX,posY);
}
```

Now, the handleClick function looks a little different to what has occurred previously because we have now taken into consideration the offset of the canvas to the edge of the border. At the moment our canvas is 50 pixels form the edge, this offset is calculated and removed to ensure that when we click on the canvas we get much more precise reading when we click.

HandleClick is then calling the activate audio function and is passing in the corrected posX and posY.

Create activateAudio like this:

```
function activateAudio(x,y)
{
    var soundToPlay = checkImage(x,y);
    switch(soundToPlay){
        case 0: createjs.Sound.play(soundID_Fire);break;
        case 1: createjs.Sound.play(soundID_Mud);break;
        case 2: createjs.Sound.play(soundID_Rain);break;
        case 3: createjs.Sound.play(soundID_Wind);break;
    }
}
```

ActivateAudio collects the x/y coordinates and then feeds them into a new function called checkImage. Once checkImage has been run, the variable soundToPlay is supplied a number, this number is then feed into the switch command to indicate which sound element to play.

We need to create checkImage now. Code this up:

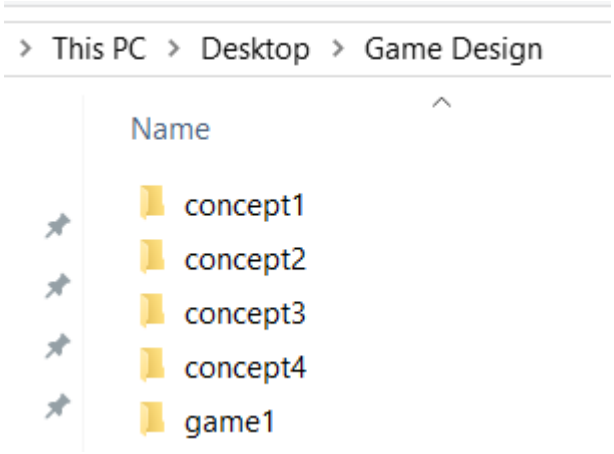
```
function checkImage(x,y)
{
    for(item=0;item<=maxStars;item++)
    {
        var startX_startWidth = starX[item];
        var startX_maxWidth = starX[item] + starSize[item];
        var startY_startHeight = starY[item];
        var startY_maxHeight = starY[item] + starSize[item];
        if((x>=startX_startWidth) && (x<=startX_maxWidth))
        {
           if((y>=startY_startHeight) && (y<=startY_maxHeight))
            {
                return item;
            }
        }
    }
}
```

CheckImage seems to be a very complicated function, but if you recall our star information is stored within an array. Therefore when we check the image, we cycle through each element in the array , determining it's box co-ordinates and then comparing the click to this box. If the box is clicked then the array position is returned so that each star can play a separate audio file.

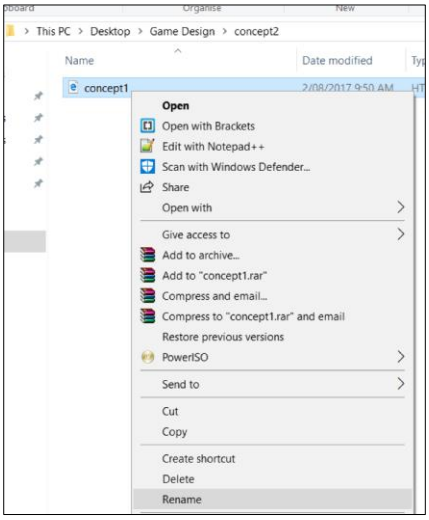
Save and test, when you click on the stars an audio file should play, irrespective on which star and based on the random position it ends up.

Game 1

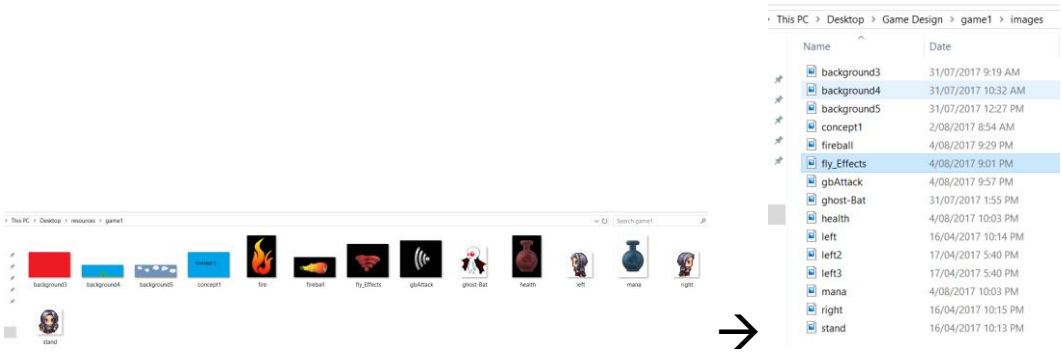
To start with, in the games design folder create a new folder called game1



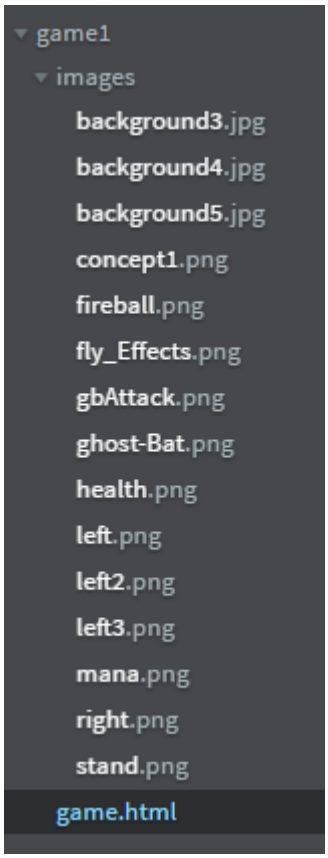
Now to speed up the coding, we can copy the concept1.html file into the game1 folder and then rename it to game.html



In addition, create the images folder and transfer the files from the resources\game1 folder to the images folder.



Then expand the game1 folder and it's images folder, you should have the following.



From here we need to change a few little parts of the code before we start.

Let's change the title from concept 1 to game1

```
<title>Game 1</title>
```

To start with let's change the background image

```
//background image
var bgImage = new Image();
bgImage.src = "images/game1.png";
bgImage.addEventListener('load',init,false);
```

Then remove the timer variable and its console line.

```
    window.setTimeout(callback, 1000 / fps);
    };
})();


function init()
```

We'll also remove the lines from the html telling us how to access the console.

```
<body>
    <canvas id="myCanvas" width="1000" height="600"
    class="canvasColour"></canvas>
    <script>
```

Next, we add the clearScreen function and drawBackground function.

Add the following global variables

```
//background image
var bgImage = new Image();
bgImage.src = "images/background4.jpg";
bgImage.addEventListener('load',init,false);
var drawX = 0;
var drawY = 0;
var fps =120;
```

Add function clearScreen()

```
function clearScreen()
{
    ctx.clearRect(0,0,gameWidth,gameHeight);
}
```

Add drawBackground()

```
function drawBackground()
{
    ctx.drawImage(bgImage,drawX,drawY);
}
```

Then we modify gameLoop

```
function gameLoop()
{
    setTimeout(function()
    {
        requestAnimFrame(init);
        clearScreen();
        drawBackground();
    },1000/fps);
}
```

If you save and test, you should see the following

Click to restart

Go Full screen

Now we will add the character to the game

Use the following code

We'll need some global variables, so type this up:

```
//Chibi
var chibi = new Image();
chibi.src = "images/stand.png";
var chibi_left = new Image();
chibi_left.src = "images/left.png";
var chibi_right = new Image();
chibi_right.src = "images/right.png";
var flyEffect = new Image();
flyEffect.src = "images/fly_Effects.png";
chibi_speed = 3;
var posx = 100;
var posy = 500;
```

Followed by the drawChibi function

```
function drawChibi()
{
    ctx.drawImage(chibi,posx,posy);
}
```

And a modification to gameLoop

```
function gameLoop()
{
    setTimeout(function()
    {
        requestAnimFrame(init);
        clearScreen();
        drawBackground();
        drawChibi();
    },1000/fps);
}
```

All of this goes together to give the following page

Click to restart

Go Full screen

So we have our starting layout, now in this game we want the player chibi figure to be controlled by keyboard, so we will modify the variables, as we will include different images; modify the drawChibi function to take into account of these images and introduce keyboard control. Of which we will control keypress up and keypress down. Each keypress will trigger a move capability for the character, such as adding or removing to the x/y co-ordinates.

Global Vars

```
//check if the keys are pushed
document.onkeydown = KeyCheckDown;

//check if the keys are no longer pushed down
document.onkeyup = KeyCheckUp;

//keyGlobal
var kLeft = false;
var kRight = false;
var kUp = false;

var moveBackground = posx;
```

Modification of drawChibi

```
function drawChibi()
{
    if (kLeft)
    {
        ctx.drawImage(chibi_left,posx,posy);
        posx = posx-chibi_speed;
        moveBackground = moveBackground - chibi_speed;
        if(posx <=1) posx = 1;
        if(posy<500) fly();
    }
    else if (kRight)
        {
            ctx.drawImage(chibi_right,posx,posy);
            posx = posx + chibi_speed;
            moveBackground = moveBackground + chibi_speed;
            if(posx>=900) posx=900;
            if(posy<500) fly();
        }
    else
    {
        ctx.drawImage(chibi,posx,posy);
        if(kUp)
            fly();
    }
}
```

Next, we add in two functions for keyboard press.

This first function activates when the key is pressed down, to ensure that when we take our fingers off the keys the action stops, we have the second function which checks for when the key is released.

```
function KeyCheckDown(e)
{
    var KeyID = (window.event) ? event.keyCode : e.keyCode;
    //console.log("The code for whichever key is pressed: ",KeyID);
    switch(KeyID)
    {
        case 65: //key a Left
            posx = posx - chibi_speed;
            kLeft = true;
        break;
        case 87: // w Up
            posy = posy - 45;
            if (posy<=0) posy = 0;
            kUp = true;
        break;
        case 68: //key d Right
            posx = posx + chibi_speed;
            kRight = true;
        break;
        case 83: // key s Down
            posy = posy + chibi_speed;
            if(posy >=500)
                posy=500;
            kUp = true;
        break;
        case 32: // key spacebar
            shoot = true;
        break;
    }
}
```

This function sets a group of variables to false when the keys are released.

```
function KeyCheckUp(e)
 {
     /*
     This function turns off the changes that are implemented when a key is pressed, this
     allows for the action of movement to stop occuring.
     */
     var KeyID = (window.event) ? event.keyCode : e.keyCode;
     console.log("The code for whichever key is pressed: ",KeyID);
     switch(KeyID)
     {
         case 65: //key a Left
             kLeft = false;
             kRight = false;
             kUp = false;
         break;
         case 87: // w Up
             kUp = false;
             kLeft = false;
             kRight = false;
         break;
         case 68: //key d Right
             kRight = false;
             kLeft = false;
             kUp = false;
         break;
         case 83: // key s Down
             kUp = false;
             kLeft = false;
             kRight = false;
         break;
         case 32: // key spacebar
             console.log("This is where we shoot",KeyID);
             shoot = true;
         break;
     }
 }
```

If you save and test, you will see that our character can ‘run’ around the canvas with no problems at all. One drawback to this is that he can float in the air with no issues. To fix this, we will introduce a function for gravity and implement the fly function which is listed in the drawChibi function

The fly function

```
function fly()
{
     ctx.drawImage(flyEffect,posx-chibi.width,posy+(chibi.height/2));
}
```

The gravity function.

```
function gravity()
{
    if(posy !=500){
        if (posy<=500){
            posy = posy+3;}
    }
    else{
        posy=500;
    }
}
```

Most of the new functions will run without modifying the gameloop, but we want gravity to be a constant in the environment, so we need to modify the function game loop like so:

```
function gameLoop()
{
    setTimeout(function()
    {
        requestAnimFrame(init);
        clearScreen();
        drawBackground();
        drawChibi();
        gravity();
    },1000/fps);
}
```

If you save and test, we have our chibi character flying around the world, but always being pulled to the ground, but he is unable to fall through the ground. In addition, we've added a fly spell effect so at least he looks good doing it. It should look like this:

Now, we have our chibi character moving around, but the world is very static, what we will do is attach some code to make the background move with our character. In addition we will also add additional background images that will feed into each other.

To start, we will add some additional global variables for images.

```
var moveBackground = posx;
var bgImage2 = new Image();
bgImage2.src="images/background5.jpg";
var bgImage3 = new Image();
bgImage3.src="images/background3.jpg";
var startPos = posx;
var drawX_2 = gameWidth*2;
var drawX_3 = gameWidth*4;
```

Then we will re-write the way that backgrounds are drawn. These backgrounds take into consideration that the character is moving and as such, use the characters movement to move the background images.

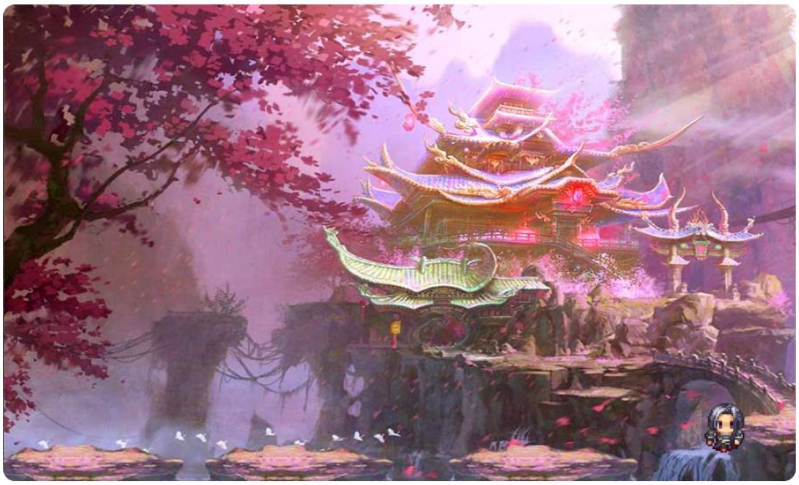
```
function drawBackground()
{
    var  moveScreen = moveBackground - startPos;

    if(moveBackground>startPos)
    {
        if((moveBackground>=900) && kRight)
        {
            ctx.drawImage(bgImage,drawX-chibi_speed,drawY);
            ctx.drawImage(bgImage2,drawX_2-chibi_speed,drawY);
            ctx.drawImage(bgImage3,drawX_3-chibi_speed,drawY);
        }
        ctx.drawImage(bgImage,drawX-moveScreen,drawY);
        ctx.drawImage(bgImage2,drawX_2-moveScreen,drawY);
        ctx.drawImage(bgImage3,drawX_3-moveScreen,drawY);
    }
    else if(moveBackground<startPos)
    {
        ctx.drawImage(bgImage,drawX+chibi_speed,drawY);
        ctx.drawImage(bgImage2,drawX_2+chibi_speed,drawY);
        ctx.drawImage(bgImage3,drawX_3+chibi_speed,drawY);
    }
    else
    {
        ctx.drawImage(bgImage,drawX,drawY);
    }
}
```

Save and test, the background images will move with the movement of the character. You should be able to move the character onto the differing backgrounds.

Background 1

Background 2

Background 3



With our character able to move through differing landscapes, the world could be extended indefinitely.

Before continuing, there is one aspect of the code that is linked up, yet not implemented, this is the full screen function, it looks like this:

```
function goFullScreen()
{
    if(canvasObject.requestFullScreen)
        canvasObject.requestFullScreen();
    else if(canvasObject.webkitRequestFullScreen)
        canvasObject.webkitRequestFullScreen();
    else if(canvasObject.mozRequestFullScreen)
        canvasObject.mozRequestFullScreen();
}
```

This code is designed to expand the canvas to the full size of the browser.

Next, we will add the fireball effect for our character. Add the following.

Global Variables

```
var fireball = new Image();
fireball.src = "images/fireball.png";
var shoot = false;
var fireballX = posx;
var fireballY = posy;
var fbCollision = false;
```

Drawing the fireball:

```
function drawFireball()
{
    var fireballSpeed = 5;
    if(fireballX<=1000)
    {
        ctx.drawImage(fireball,fireballX,fireballY,100,48);
        fireballX = fireballX + fireballSpeed;
    }else
    {
        shoot = false;
    }
}
```

Then we need to modify the keyDown function, this is so the fireball will leave from the character once the spacebar is pressed.

```
      case 32: // key spacebar
          shoot = true;
          fireballX = posx;
          fireballY = posy;
      break;
  }
```

And lastly for the fireball, we modify the gameloop

```
function gameLoop()
{
    setTimeout(function()
    {
        requestAnimFrame(init);
        clearScreen();
        drawBackground();
        drawChibi();
        if(shoot)
            drawFireball();
        gravity();
    },1000/fps);
}
```

When you save and test, you should see something like the following

Now we will bring in an NPC for the chibi, this ghost bat will follow the chibi, so to start off with, we will need some global variables.

```
//Ghost Bat
var gBat = new Image();
gBat.src = "images/ghost-Bat.png";
var ghostBatX = 1200;
var ghostBatY = 0;
```

As you can tell, ghost bat starts off the screen, so to make him more useful, we will add movement to him when we draw him to the screen.

Draw ghostbat

```
function ghostBat()
{
    ctx.drawImage(gBat, ghostBatX,ghostBatY);
    moveGhostBat();
}
```

MoveGhostbat

```
function moveGhostBat()
{
    var speed = 1;
    if(ghostBatX>= posx){
        ghostBatX = ghostBatX - speed;
    }else if(ghostBatX<= posx){
        ghostBatX = posx + speed;
    }
    if(ghostBatY>= posy){
        ghostBatY = ghostBatY - speed;
    }else if(ghostBatY<= posy){
        ghostBatY = posy + speed;
    }
}
```

Modify gameloop

```
function gameLoop()
{
    setTimeout(function()
    {
        requestAnimFrame(init);
        clearScreen();
        drawBackground();
        drawChibi();
        if(shoot)
            drawFireball();
        ghostBat();
        gravity();
    },1000/fps);
}
```

Once this is done, save and test. You should see the following

There is a lot that can be done to the game; elements such as health, mana can be added to the character as well as collisions for the fireball and ghostbat.