Carleton University - School of Computer Science Honours Project

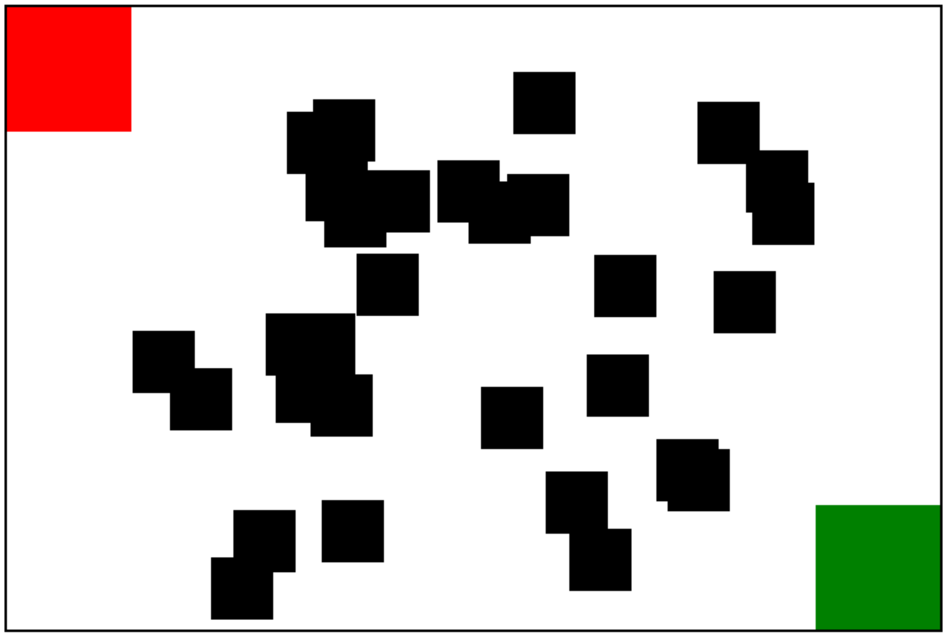
Winter 2015

2D Gaming Tutorials using JavaScript and HTML5

By: Jordon Lyn

Supervisor: Louis D.Nel (LD Nel - School of Computer Science)

Date: April 4th, 2015



ABSTRACT

Although DirectX is a common and popular tool to use in the world of computer game development, many game developers have started to move away from DirectX and use other computer programming languages for developing computer games. For my honours project, I have decided to create gaming tutorials for the course known as COMP 2501: Computer Game Design and Development taught at Carleton University using JavaScript and HTML5 instead of DirectX. These tutorials, while focusing on 2D gaming, will deal with many of the major topics taught in computer game development, such as collision detection, movement, animation, and rotation. The purpose of these tutorials is to help future students of COMP 2501 understand these topics better, and motivate them to explore many other creative ideas in this line of work rather than focus on DirectX.

ACKNOWLEDGEMENTS

I give acknowledgement to the staff and Teaching Assistants of Carleton University's School of Computer Science for helping me understand various concepts and topics of computer science and for making my four years in Computer Game Dev an exciting and well-enjoyed experience.

I give acknowledgement to my fellow classmates in computer game dev for helping me with the various assignments and projects.

I give acknowledgement to the authors of articles that I will use as references in this report.

I acknowledge the creators of the website http://w3schools.com for its JavaScript tutorials which helped me understand many of the JavaScript functions and how they work.

I acknowledge my family for always giving me the support and belief that helped me persevere in difficult times.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

MOTIVATION

When Microsoft released Windows 95 to the public in August of 1995, three employees of the company, Craig Eisler, Alex St. John, and Eric Engstrum, noticed that most programmers preferred working on the Microsoft gaming platform known as MS-DOS, rather than use the new operating system[1]. This was due to the fact that because of its protected memory model, Windows 95 no longer gave programmers direct access to video cards, mouse, keyboards, and sound devices (and the rest of the system), something that MS-DOS did[2]. Eisler, St. John, and Engstrum decided to resolve this problem by developing the first version of DirectX upon the concepts of another development system known as "Exodus", developed by Kinesoft Development, a company that Microsoft worked closely with[3]. In September of 1995, DirectX 1.0 was released to the public, although this version was given the name "Windows Games SDK"[4]. It was because of this that Microsoft came to realize that games in fact play a key role in the user's choice of an operating system[5], which is why they release newer versions of DirectX per year (see Table 1 to view different versions of DirectX).

[1] "The History of DirectX", CodingUnit, 8 April 2015, http://www.codingunit.com/the-history-of-directx
[2] Ibid
[3] Ibid
[4] Ibid
[5] Rosen David, "Why you should use OpenGL and not DirectX", Wolfire Blog Games, 8 Jan 2010, 8 April 2015, http://blog.wolfire.com/2010/01/Why-you-should-use-OpenGL-and-not-DirectX

FIGURE 1: DirectX Logos over the years
(Left) - The DirectX logo from 1995 to 1999
(Centre) - The DirectX logo from 1999 to 2010
(Right) - The current DirectX logo
http://logos.wikia.com/wiki/DirectX

DirectX Version	Version Number
DirectX 1.0	4.02.0095
DirectX 2.0 and 2.0a	4.03.00.1096
DirectX 3.0 and 3.0a	4.04.0068/69
DirectX 4.0	Version Number was never released
DirectX 5.0 (Was later released with Windows 98)	4.05.00.0155 4.05.01.1721/1998 (Windows 98)
DirectX 6.0	4.06.02.0436
DirectX 7.0	4.07.00.0700
DirectX 7.0a	4.07.00.0716
DirectX 8.0	4.08.00.0400
DirectX 8.1	4.08.01.0810 4.08.01.0881
DirectX 9.0	4.09.0000.0900
DirectX 9.0a	4.09.0000.0901
DirectX 9.0b	4.09.0000.0902
DirectX 9.0c	4.09.0000.0904
DirectX 10	6.00.6000.16386
DirectX 10 under SP1 in Vista and Windows Server 2008	6.00.6001.18000

TABLE 1 - Different Versions of DirectX over the years
Version number is also included
http://www.computerhope.com/jargon/d/directx.htm

Although initially a failure when Microsoft entered the gaming market in 2001 with the original Xbox (losing over $4 billion in the process), it did however set the stage for Microsoft to dominate the gaming market in the next generation as many major PC games today use DirectX and run on both Windows and Xbox

360[6]. Seeing this, it makes us wonder why programmers choose DirectX? The reason being is because of the positive feedback loop in choosing an API in game development shifted in favour of DirectX in 2005[7]. This positive feedback loop for DirectX is because of three main reasons: network effects and vicious cycles, the FUD campaign against OpenGL, and misleading marketing campaigns.

Network effects state that popular APIs get better support from graphic card vendors, and programmers are more likely to already know how to use it, creating a vicious cycle where more gaming projects will be developed using DirectX leading to more programmers needing to learn DirectX making it cheaper to learn DirectX than OpenGL[8]. The FUD (fear, uncertainty, and doubt) campaign that Microsoft initiated about the future of OpenGL around the release of Windows Vista, caused panic within the OpenGL community, leading many programmers to switch from OpenGL to DirectX[9]. The misleading marketing campaigns launched by Microsoft exaggerated the merits of DirectX and convinced gamers that DirectX updates are the only way to access the latest graphic features; however gaming journalists proved that is fact no difference and that the features in the updates can work on the previous version by simply tweaking a configuration file[10] (See Figure 2 for example).

[6] Ibid
[7] Ibid
[8] Ibid
[9] Ibid
[10] Ibid

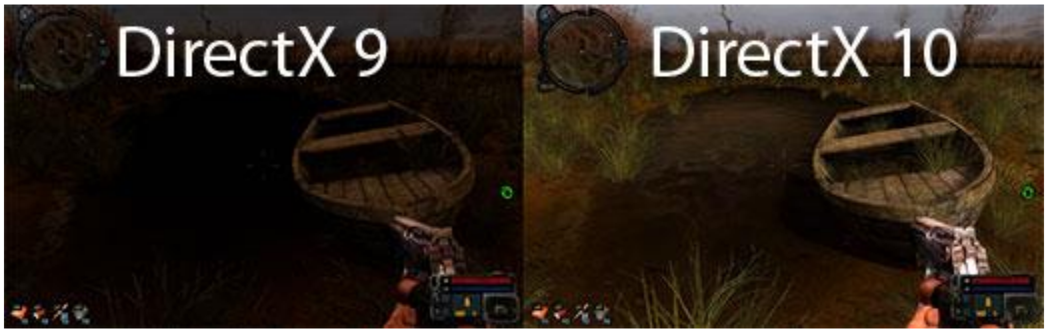


FIGURE 2: DirectX Comparison.
Comparison of the game "Stalker: Call of Pripyat" on DirectX 9 and DirectX10. Microsoft initially led people to think that switching from 9 to 10 would magically transform the graphics from stupidly dark to normal.
http://blog.wolfire.com/2010/01/Why-you-should-use-OpenGL-and-not-DirectX

However, despite the popularity of DirectX and considering the vast majority of programmers that use it, there are still a few programmers out there who prefer to use other programming languages, such as OpenGL, instead of DirectX. Even though OpenGL no longer is used in games and faces constant attacks by Microsoft, people still use OpenGL instead DirectX because of reasons described in Table 2[11].

Reason	Description
OpenGL is more powerful than DirectX	• OpenGL draw calls are faster than DirectX draw calls • Direct access to all new graphic features on all platforms (DirectX only gives snapshots on latest version on Windows)
Cross-platform	• Majority of PC gamers today still use Windows XP (DirectX 10 and 11 do not work on XP) • OpenGL is the only way to deliver the latest graphics to XP gamers
Better for the future of gaming	• Microsoft monopoly on gaming could lead to bad future for gamers and game developers

TABLE 2: Reasons people still use OpenGL

[11] Ibid

For my honours project, I decided to do something similar and create gaming tutorials outside of DirectX, but instead of using OpenGL, I decided to use JavaScript and HTML5. The reason I chose JavaScript and HTML5 over OpenGL is because I only recently learned how to use OpenGL and felt it would be much easier and less stressful for me to use in JavaScript than in OpenGL. JavaScript and HTML are two of the three mandatory languages that a programmer must learn (the third being CSS), as they are both used to respectively program the behaviour of a web page and define its content[12]. Although multi-threaded work in DirectX and OpenGL can give out more desired results than JavaScript and HTML, a clear advantage JavaScript and HTML have is that the code written with these two languages is more portable to other platforms that have HTML-based options, such as the Internet; however, it does require that the code made specifically for Windows is to be kept isolated, which is possible[13]. My motivation for doing this project is that if I can create something outside of DirectX, then perhaps it will show future students of COMP 2501 that creativity and knowledge in game development is not just restricted to DirectX.

[12] "JavaScript Tutorials", w3schools.com, 9 April 2015, http://w3schools.com/js/default.asp

[13] Brockschmidt Kraig, "My Take on HTML/JS vs. C#/XAML vs. C++/DirectX (choosing a language for a Windows Store app)", kraig brockschmidt, 17 Jan 2013, 9 April 2015, http://www.kraigbrockschmidt.com/2013/01/17/html-javascript-xaml-directx-language-windows-store-app/

METHODOLOGY (Introduction)

The original goal for this project was to make a total of seven gaming tutorials for each of the following topics: basic display, movement, rotation, animation, collision detection, transparency, and sound. However as time progressed, I decided that the work I did for the first three topics was not sufficient enough; so I decided to create an extra tutorial for each, adding three more to my original seven for a total of ten. Following that decision, I also decided that it would be better to simply implement sound into collision detection, so I discarded the sound tutorial completely, meaning my finalized honours project should have a total of nine tutorials. For all topics except for movement and collision detection, I would use pictures I found online and used them for assistance in the tutorials (For collision detection I use wav files to play sound). The next part of this report will speak about each of the topics I have used for this project, and the methods I took before coming up with the final product.

METHODOLOGY (Basic Display)

Before I began working on the tutorials, I decided that the first thing I needed to do was to get myself familiar with JavaScript. I figured the best way to do so was by showing the basic display of a square with an image inside it. The first thing I needed was the HTML <canvas> element which is used to draw graphics on a web via scripting, usually via JavaScript[14] (See Table 3 to view browser support for the <canvas> element).

Browser	<canvas>
Google Chrome	4.0
Internet Explorer	9.0
Firefox	2.0
Safari	3.1
Opera	9.0

TABLE 3: Browser support for the <canvas> element.
The numbers indicate the first browser version that fully supports the <canvas> element
http://www.w3schools.com/html/html5_canvas.asp

When defining a canvas, it is required that width and height attributes are specified to define the size of the canvas along with an id attribute to be later used in a script that draws content onto the canvas; the style attribute is not required, but is recommended to add a canvas border[15] (See Figure 3 for my example).

```
<canvas id = "Honours Project 1" width="310" height="310"
style="border:2px solid #000000;">
Your browser does not support HTML5 canvas.
</canvas>
```

APPENDIX 1: Defining a canvas in JavaScript.

[14] "HTML5 Canvas", w3schools.com, 10 April 2015, http://www.w3schools.com/html/html5_canvas.asp
[15] Ibid

Before we can begin drawing on the canvas, we need to declare the variables that will be used to help us with the drawing. The first step in doing so is to retrieve the canvas element with the HTML DOM method getElementById(). After that, define a drawing object for the canvas with the getContext() method, which is a built-in HTML object, with properties and methods for drawing[16]. The next thing I did was define a variable for the image using the newImage() method which takes in width and height attributes to define the size of the image. Another thing that is needed in defining a new image is to load the image source to the image variable (See Figure 4 for example).

```
var canvas = document.getElementById("Honours Project 1");
var tut1 = canvas.getContext("2d");
var image = new Image(250, 250);
image.src = "game-design.jpg";
```

APPENDIX 2: Defining the variables for drawing in JavaScript.

Now we can begin drawing on the canvas. The first step is drawing the square which can be done in many numerous ways; I chose to draw my square with the fillRect() method which draws a "filled" rectangle with black being the default color[17]. This syntax of all rectangle methods in JavaScript take in four variables with the first two being the x and y coordinates for the left top corner of the rectangle and the last two being the width and height. Once then, I then called the drawImage method to show the image on the square. The syntax of drawImage() is similar to all rectangle methods, with the only difference being is it takes five variables with the first one defining the image it is supposed to draw.

[16] "HTML Canvas Drawing", w3schools.com, 10 April 2015, http://www.w3schools.com/canvas/canvas_drawing.asp
[17] "HTML canvas fillRect() method", w3schools.com, 10 April 2015, http://www.w3schools.com/tags/canvas_fillrect.asp

```
tut1.fillRect(10, 10, 290, 290);
tut1.drawImage(image, 30, 30, 250, 250);
```

APPENDIX 3: Drawing the square and the image
It is recommended to call the fillRect method first otherwise it will make the rectangle cover the image

Final Result:

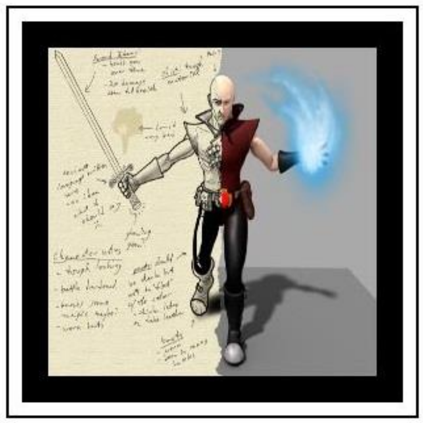


FIGURE 3: Finished Product for Basic Display

For the extra tutorial I made for this topic, I drew multiple squares, each having a different color and a different image which I took from an image array I created.

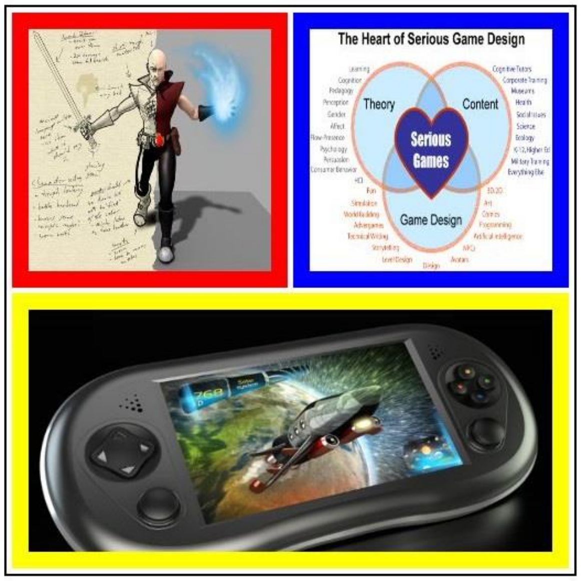


FIGURE 4: Basic Display for Multiple Squares and Images

METHODOLOGY (Movement)

The next section I worked on for my honours project was giving movement to my game objects because when playing a game, gamers would like to move their character around to explore the environment hoping to progress further into the story. If this was not featured in a game, then the game becomes pointless and boring, leading gamers to steer away from your game. The original plan was to only draw a square and move it within the region of the canvas using the arrow keys; however I decided to be more elaborate with this section and add in a circle and have it move with the WASD keys. To be more organized, I created two draw functions to draw the square and circle respectively. To draw a circle, you need to call the beginPath() method to begin a path or reset the current path[18], then call the arc() function to create the circle and either the stroke() or fill() methods to have the circle actually appear on the canvas (Not having the beginPath() method could cause some errors).

```
var square_x = 10;
var square_y = 10;
var circle_x = 500;
var circle_y = 420;

drawSquare();
drawCircle();

function drawSquare(){
  tut2.fillStyle = 'red';
  tut2.fillRect(square_x , square_y, 100, 100);
}
function drawCircle(){
  tut2.fillStyle = 'blue';
  tut2.beginPath();
  tut2.arc(circle_x, circle_y, 50, 0, Math.PI * 2, false);
```

[18] "HTML canvas beginPath() method", w3schools.com, 11 April 2015, http://www.w3schools.com/tags/canvas_beginpath.asp

```
  tut2.fill();
}
```

APPENDIX 4: Drawing the square and the circle
The syntax of the arc function takes in five variables:
(The x and y coordinates for the centre of the circle, the radius of the circle, the start and end angles and an optional boolean value which determines if the circle should be drawn counterclockwise or clockwise)
(Drawing a full circle requires the end angle to be 2 * Math.PI)

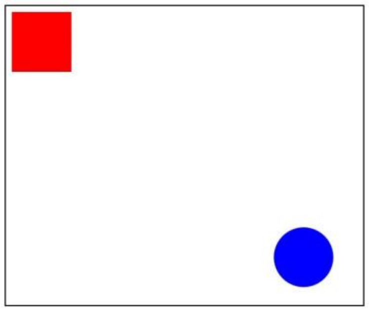


FIGURE 5: Objects for Movement

To apply certain events to objects such as movement, an event handler is required so that when that certain event occurs, the handler makes it easier for the object to react to that event. The syntax of the addEventListener() method takes in three variables: the first variable is the type of event that is to occur, the second variable being is the function that is to be called when the event happens, and the third variable being an optional boolean value which is to used to specify whether to use event capture or event bubbling[19] (See Table 4 for browser support for the EventListener method[20]). For both the square and circle objects, I created two functions that move either one of the objects depending on which type of key is pressed. How the functions operate is that when a certain key on the keyboard is pressed, the functions takes the keyCode property and returns the Unicode character code of the key that is pressed[21]. If the character code

[19] "JavaScript HTML DOM EventListener", w3schools.com, 11 April 2015, http://www.w3schools.com/js/js_htmldom_eventlistener.asp
[20] Ibid
[21] "KeyboardEvent keyCode Property", w3schools.com, 11 April 2015, http://www.w3schools.com/jsref/event_key_keycode.asp

matches to one of the correct values, then it will move the square or circle based on the direction I had given that key, provided that the object does not leave the canvas.

Browser	addEventListener()	removeEventListener()
Google Chrome	1.0	1.0
Internet Explorer	9.0	9.0
Firefox	1.0	1.0
Safari	1.0	1.0
Opera	7.0	7.0

TABLE 4: Browser support for the EventListener() methods
For earlier versions of Internet Explorer and Opera, use the attachEvent and detachEvent methods

```
function SquareMovement(event){
 //Move left using the Left arrow Key
 if(event.keyCode == '37'){
   if(square_x  > 0)
     square_x  -= 10;
 }
 //Move up using the Up arrow Key
 if(event.keyCode == '38'){
   if(square_y > 0)
     square_y -= 10;
 }
 //Move right using the Right arrow Key
 if(event.keyCode == '39'){
   if(square_x + 100 < c.width)
     square_x  += 10;
 }
 //Move down using the Down arrow Key
 if(event.keyCode == '40'){
   if(square_y + 100 < c.height)
     square_y += 10;
 }
 //Reset everything using the 'R' key or 'r' key
 if((event.keyCode == '82') || (event.keyCode == '114')){
   square_x  = 10;
   square_y = 10;
 }
 c.width = c.width; //This clears out the canvas when the square is
moving
 drawSquare();
 drawCircle();
}
window.addEventListener("keydown", SquareMovement);
```

APPENDIX 5: Event Listener for the square

The event "keydown" means whenever a key is pressed
The line "c.width = c.width" is needed because without it, a trail gets left behind.
The two functions to draw both objects are called because without them, the objects disappear everytime an arrow key is pressed
The event listener for moving the circle is similar except it uses the WASD keys

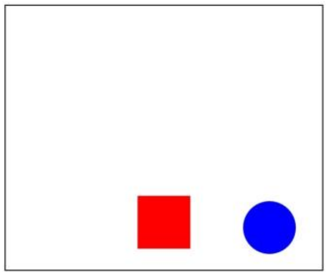


FIGURE 6: The square has been moved.

For the extra tutorial I made for this topic, my plan was to draw a square and make it move with the mouse. This is done by taking the clientX and clientY properties and returning the coordinate of the mouse pointer while the mouse is moving through the canvas and having the square follow it. I tried to do something different from the addEventListener() method by declaring an onmousemove event handler when defining the canvas element; but changed my mind and used the addEventListener() method, but instead of the keyboard I use the mouse.

```
function MoveSquare(event){
  if(event.clientX < c.width && event.clientY < c.height){
    x = event.clientX;
    y = event.clientY;
  }

  if(event.clientX > c.width || event.clientY > c.height){
    x = 5;
    y = 5;
  }
  c.width = c.width;
  tut25.fillRect(x, y, 150, 150);
}
window.addEventListener("mousemove", MoveSquare);
```

APPENDIX 6: Moving an object with the mouse
The event "mousemove" is when the mouse is moving while the pointer is over the canvas

METHODOLOGY (Rotation)

Rotation is when an object performs a circular movement around a point of interest. Although rotation is better implemented in 3D gaming, with yaw (rotation around the Y-axis), pitch(rotation around the X-axis), and roll (rotation around the Z-axis), it still can be used in 2D gaming (e.g. changing the direction the player is facing when trying to move forward or backward). My plan for this tutorial was to take an image and make the image rotate 30 degrees counterclockwise or clockwise using the left and right arrow keys respectively. In order to correctly rotate an object, you must first translate it to the point of interest, call the rotation, then undo the translation. I accomplished this using the translate() and rotate() methods (all browsers support both methods), but before I did any of this, I created a clear rectangle over the canvas so that residual imaging would not be a problem.

```
function Rotate(event){
//To rotate an object translate to the point of interest
//call the rotation then translate it back
//Create a clear rectangle over the canvas so that residual imaging is
not a problem
  tut3.clearRect(0, 0, canvas.width, canvas.height);

//Left arrow key makes image go counter-clockwise
  if(event.keyCode == '37'){
    tut3.translate(160, 150);
    tut3.rotate(-30*Math.PI/180);
    tut3.translate(-160, -150);
  }

//Right arrow key makes image go clockwise
  if(event.keyCode == '39'){
    tut3.translate(160, 150);
    tut3.rotate(30*Math.PI/180);
    tut3.translate(-160, -150);
  }
  tut3.drawImage(games, 160, 150, 160, 160);
}

window.addEventListener("keydown", Rotate);
```

APPENDIX 7: Rotating an Image
Rotations are done in radians in JavaScript, hence Math.PI/180 is used to convert the degrees into radians

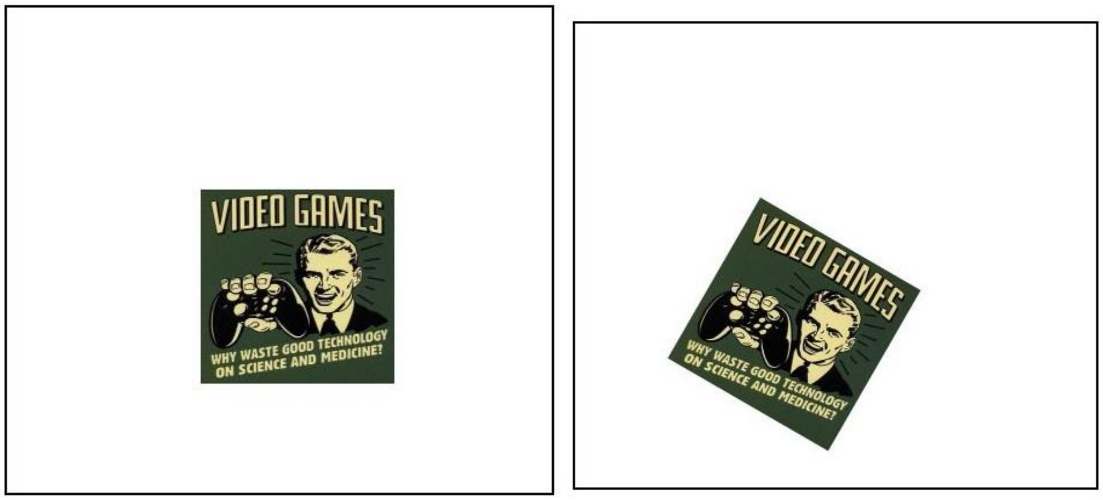

VIDEO GAMES
WHY WASTE GOOD TECHNOLOGY ON SCIENCE AND MEDICINE?
VIDEO GAMES
WHY WASTE GOOD TECHNOLOGY ON SCIENCE AND MEDICINE?

FIGURE 7: Before and after a 30-degrees counterclockwise rotation
Clockwise rotation can achieve same thing with a 330-degree rotation

For the extra tutorial I made for this topic, I made the image rotate clockwise by clicking the mouse button. Originally, I created two button onclick functions that would make the image rotate by clicking on the button, and although it worked, I decided to change it to work with a mouse event handler. I had first tried a mousemove event handler but all that did was make the image rotate completely out of control, so I then chose the click event handler instead, and it works much better.

```
function Rotate(){
  tut35.clearRect(0, 0, canvas.width, canvas.height);
  tut35.translate(190, 150);
  tut35.rotate(30 * Math.PI/180);
  tut35.translate(-190, -150);
  tut35.drawImage(games, 190, 150, 180, 180);
}
window.addEventListener("click", Rotate);
```

APPENDIX 8: Rotate an Image with the mouse

METHODOLOGY (Animation)

Animation is the idea of taking a series of images greatly similar to one another and by viewing them in a rapid state to create motion within the images. Animation is used in many different ways, such as flip books, movies, tv shows, etc. and can be done in many different styles and techniques. Such examples are traditional animation, which has each image drawn by hand (e.g. classic animated movies), and stop-motion animation where real-life objects are manipulated and photographed frame by frame to create motion (e.g. claymation). While both types of animation are still around, traditional animation has been replaced by computer animation, as it deals with both 2D and 3D animation digitally instead of by hand which makes more easier than traditional (e.g. CGI movies).

FIGURE 8: Different Animation Styles
Left - Traditional Animation
Centre - Stop-Motion Animation
Right - Computer Animation
http://www.skwigly.co.uk/2d-or-not-2d-the-disney-feature-animation-legacy/
http://filmjunk.com/2008/11/20/neil-gaimans-stop-motion-coraline-trailer-directed-by-henry-selick/
http://www.examiner.com/article/pixar-animation-promotes-brave-during-concert-to-celebrate-their-movie-scores

The animation tutorial takes the different faces of a clock and flips through them to make it look like the hour hand is moving. The first thing I did was take all the clock images I obtained and put them into an image array, so that traversing through the images would become much easier. Constructing arrays in

JavaScript can be done in many ways; for my tutorial, I chose the newArray() method (See Appendix 8 for example).

```
var clockFaces = new Array(11);
clockFaces[0] = new Image(310, 310);
clockFaces[0].src = "12 o'clock.png";

clockFaces[1] = new Image(310, 310);
clockFaces[1].src = "1 o'clock.png";

clockFaces[2] = new Image(310, 310);
clockFaces[2].src = "2 o'clock.png";
```

APPENDIX 9: Declaring a new array and its first three elements
Arrays always start with the index 0, which is why the number put in this function during its declaration is always one less than the total length of the array

Following the declaration of the new array and defining each array element to be one of the clock faces, I then made two new variables: one is called "currentTime" which displays the current time drawn on the canvas, and the other is called "counter" which is used in the animation part of the code. Originally I had done the animation by creating two different functions, one making the clock go backward and the other making it go forward, and gave each one a keydown event handler. I recently however modified the code and merged both functions into one to make the code more organized and it still works the same way as my original format did (See Appendix 9 for new format).

```
//Change the hour of the clock
function ClockHour(event){

  //Clock goes backward
  if(event.keyCode == '37' && counter == 0)
    counter = clockFaces.length;
  if(event.keyCode == '37' && counter > 0)
    counter -= 1;

  //Clock goes forward
  if(event.keyCode == '39' && counter < clockFaces.length)
    counter += 1;
  if(event.keyCode == '39' && counter == clockFaces.length)
    counter = 0;
```

```
   currentTime = clockFaces[counter];
   tut4.drawImage(currentTime, 0, 0, canvas.width, canvas.height);
}

window.addEventListener("keydown", ClockHour);
```

APPENDIX 10: Animating the clock
I added 'if' statements within the function to deal with cases when the counter variable reached either the end or the start of the array

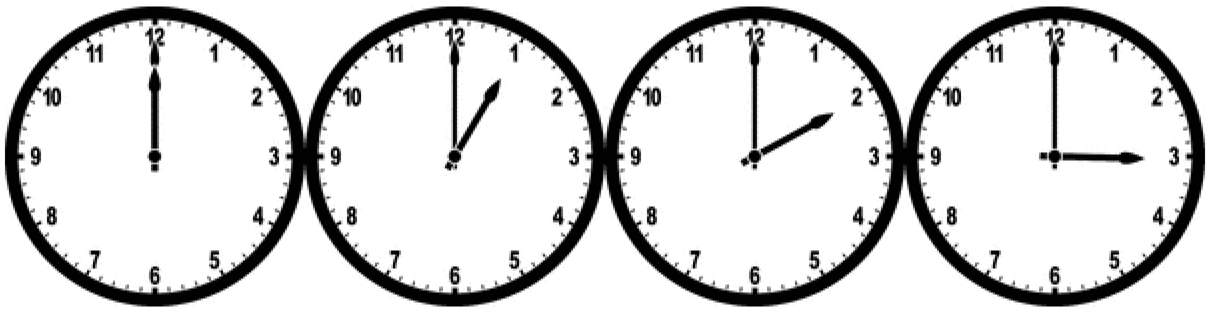


FIGURE 9: Animation in progress
With the 'ClockHour' function the user can now traverse through the different hours of the clock
Holding down either the left or right arrow keys make the hour hand spin out of control

METHODOLOGY (Collision Detection)

Collision detection deals with the situation of two objects coming into contact with one another. Out of all the topics I did in for this project, I believe that collision detection is the most important topic to be focused on, because collision detection can help change the dynamic of a game (e.g. when the player comes in contact with an enemy, collision detection is used to make the player either lose health or cause a game over and return the player to the last saved checkpoint). For this tutorial, I designed a simple spaceship game where you are traversing through an asteroid field in a severely damaged ship and trying to reach the base while avoiding any of the asteroids positioned throughout the field. The reason I chose this example is because it appears to be a popular and easy example for collision detection.

I used most of the code I created in the movement tutorial to help with the construction of this tutorial. I drew both the ship and the base just like how I drew the square in the movement tutorial. I even reused my SquareMovement() function, but added in an 'if' statement which only allows the player to move the ship only if the game is still in progress (i.e. collision has not been detected). The hardest part however was creating multiple asteroids around the field. When I first attempted this, I created two random variables that will act as the x and y positions for the asteroid, then I created a for loop in the drawAsteroids() function I designed, thinking it would display the multiple asteroids. However, when I tested it out, only one asteroid was drawn. The reason for this is because the random variables I created only accounts for one asteroid; so to fix this I created two arrays to store the x and y values for each asteroid (See Appendix 10 to view code).

```
var asteroidsX = new Array(29); //x-values for the asteroids
var asteroidsY = new Array(29); //y-values for the asteroids
for(var i = 0; i <= 30; i++){
  asteroidsX[i] = Math.floor((Math.random() * 500) + 100); //random x-
positions
  asteroidsY[i] = Math.floor((Math.random() * 400) + 50); //random y-
positions
}

//Create the asteroids that you want to avoid
//Based on example found at https://msdn.microsoft.com/en-
us/library/ie/gg589497(v=vs.85).aspx

function drawAsteroids(){
  for(var i = 0; i <= 30; i++){
    tut5.fillStyle = 'black';
    tut5.fillRect(asteroidsX[i], asteroidsY[i], asteroidWidth,
asteroidHeight);
  }
}
```

APPENDIX 11: Drawing the asteroids
The first random function originally gives a x-position between 100 and 500
The second random function gives a y-position between 50 and 400

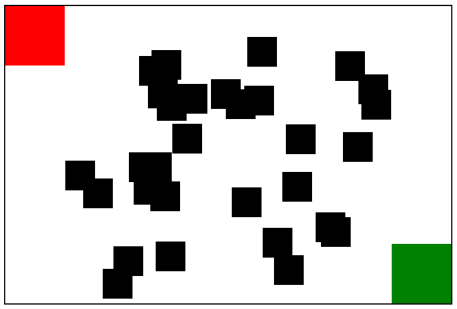


FIGURE 10: Starting the game
Objective of the game is to guide the ship (red square) to the base (green square) while avoiding the asteroids (black squares)

I did want to give the ship a laser which you could fire in order to destroy the asteroids to make reaching the base easier. However due to time constraints, I changed my mind and decided that to not add the laser, but instead add a shrink ray which allows you to shrink the ship to get into tight spaces you could not originally get into. This helps add to the story that the ship has suffered so much damage, that all weapons cannot be used.

Now we can add collision detection to the game which is dealt with in two different cases: collision between the ship and the asteroids and collision between the ship and the base. For collision detection between the ship and the asteroids, I used the basic axis-aligned bound box collision (a.k.a rectangle-to-rectangle collision given that there is no rotation) that checks for a gap between any of the four sides of the ship and an asteroid[22] (See Appendix 11). If there is no gap between a ship and an asteroid, then there is a collision and calls for an immediate game over. For the collision detection between the ship and the base, I did it much differently from the axis-aligned bound box collision. What I did was for there to be collision between the ship and the base, the ship had to be over the base and completely cover it; only then would there be a collision and the game would indicate the player they won the game (See Appendix 11).

```
//Check for collision
function CollisionDetection(){
  //Ship to Asteroid Collision (Game Over)
  for(var i = 0; i <= 30; i++){
    if((shipX < asteroidsX[i] + asteroidWidth) &&
      (shipX + shipWidth > asteroidsX[i]) &&
      (shipY < asteroidsY[i] + asteroidHeight) &&
      (shipY + shipHeight > asteroidsY[i])){
      shipHit = 1;
    }
  }

  if(shipHit){
    explosion.play();
    alive = 0;
    shipWidth = shipHeight = 0;
    asteroidWidth = asteroidHeight = 0;
  }
```

[22] "2D Collision Detection", Mozilla Developer Network, 14 April 2015, https://developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection

```
  if(!alive){
    gameover.play();
    tut5.font = "bold 36px Arial";
    tut5.fillText("Game Over!!", 450, 450);
    tut5.font = "bold 20px Arial";
    tut5.fillText("Press R or r to restart", 450, 480);
  }

  //Ship reaches the end of the base (You win the game)
  if((shipX == 650) && (shipY == 400)
     && (shipX + shipWidth == 750) && (shipY + shipHeight == 500)){
    victory = 1;
  }

  if(victory){
    winSound.play();
    asteroidWidth = asteroidHeight = 0;
    tut5.font = "bold 36px Arial";
    tut5.fillText("You Win!!", 450, 450);
    tut5.font = "bold 20px Arial";
    tut5.fillText("Press R or r to restart", 450, 480);
  }
}
```

APPENDIX 12: Collision Detection function
Checks for both conditions of collision in the game and flips the boolean values 'alive' or 'victory' when a collision occurs

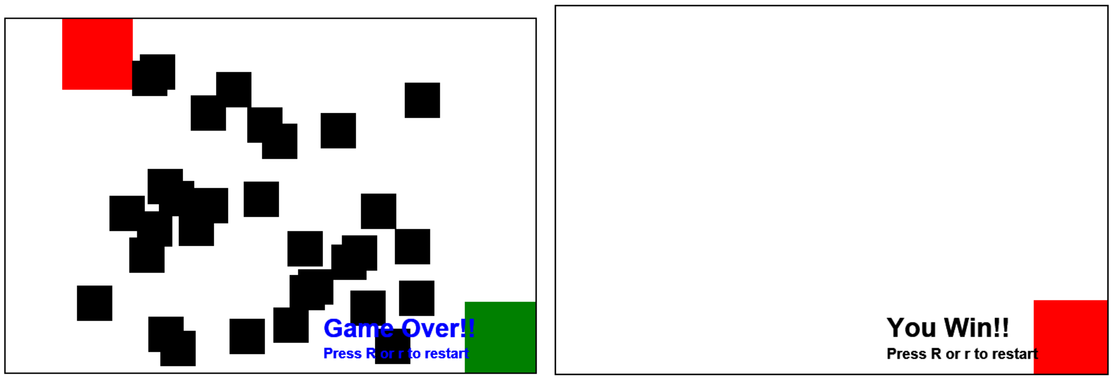


FIGURE 11: Both Cases of Collision Detection
Left - collision between the ship and an asteroid
Right - collision between the ship and the base

As an added bonus, I added sound which plays during the game, and when anyone of the collision detections occur. I did this by taking audio samples I found online, define variables for them using the new Audio() method and called the play() method to play the sound files whenever needed (Note: While all browsers can use the play() method, only one of the sounds I used in my code can be played on Internet Explorer, the rest need to be played on other browsers in order to hear them) (See Table 5 to see which sound files can be played on which browser[23]).

Browser	MP3	Wav	Ogg
Internet Explorer	YES	NO	NO
Google Chrome	YES	YES	YES
Firefox	YES	YES	YES
Safari	YES	YES	NO
Opera	YES	YES	YES

TABLE 5: Browser support for HTML5 Audio

[23] "HTML5 Audio", w3schools.com, 14 April 2015, http://www.w3schools.com/html/html5_audio.asp

METHODOLOGY (Transparency)

The last tutorial in my honours project deals with the topic transparency. Transparency deals with how well an image can be seen. This can be used in gaming to help make objects either transparent (see-through) or opaque (solid). For this tutorial, I based it off an example I found online and took two pictures I had previously used in my basic display tutorial and start them with full opacity, but every time the mouse hovers over one of the images, then that image becomes transparent. The way the code works is that it takes the opacity property and gives it a value between 0.0 and 1.0; the lower the value, the more transparent the image will be[24] (See Appendix 12 to view code and Figure 12 for example).

```
img {
   opacity: 1.0;
}

img:hover {
   opacity: 0.3;
}
```

APPENDIX 13: Transparency Code
This code shows that all images start off with opacity of 1.0 (full opacity), but everytime the mouse hovers over an image, then that the opacity of the image decreases to 0.3 (30% opacity), making it transparent.

[24] "CSS Image Opacity/Transparency", w3schools.com, 14 April 2015, http://www.w3schools.com/css/css_image_transparency.asp

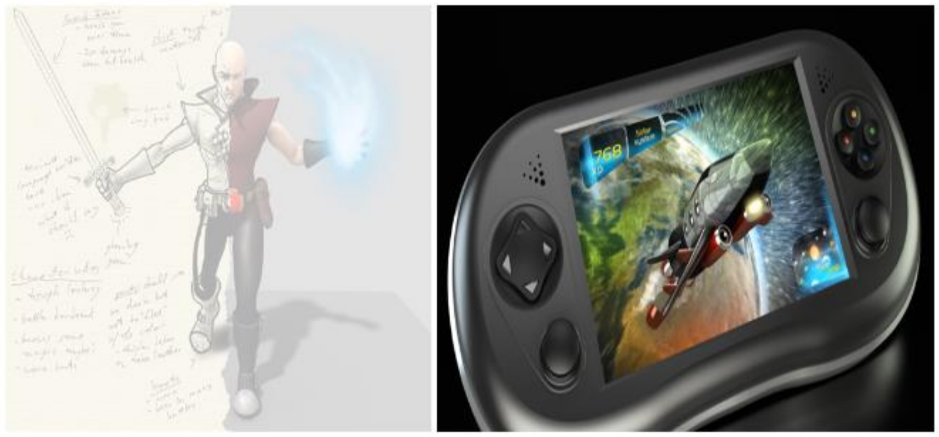


FIGURE 12: Transparency Example
As shown in this example using the code from Appendix 12, when I hover of the image on the left, its opacity decreases to 0.3 making it 70% transparent, while the image on the right still has full opacity

RESULTS

What I have achieved in this project is I not only understand the topics I used for this project better, but now I know how to construct functioning 2D gaming tutorials for each of them using a programming language that is not DirectX. One thing that works in this project is that although there may exist a few flaws, the tutorials work with no bugs, errors or crashes. Another thing that works is the event handlers I created to help my tutorials on movement, animation, and rotation work and cause no additional problems. One thing that does not work is that one of the sound files that I intended to use as the stage music in the collision detection tutorial does not work that I wanted to. It keeps playing even after a collision happens, but it does not repeat itself after it is finished playing and the only way to repeat after it is finished is to restart the game. Another thing that does not work is every time I open one of the tutorials, except for transparency, all the content that is suppose to be drawn on the canvas is blocked and there is a message at the bottom of the screen telling me that the page has been restricted from running scripts or ActiveX controls. This means that in order to see my work, I always have to click the button that allows me to view the blocked content (Note: This message will not reappear if I allow the blocked content and refresh the webpage). Working on this project really helped me explore 2D gaming outside of DirectX. JavaScript and HTML5 had hundreds of methods and tons of ideas for me to work on the tutorials and express my creativity, showing that DirectX is not the only programming language for game development.

CONCLUSION

What I liked about working on this project was that it gave me the opportunity to test myself and see how much I have learned about game development in these last four years. From what I have seen in my progress and the finalized tutorials, I feel very proud in what I have accomplished. I have learned many different concepts of game development and how each one is important to game development. But most importantly, this project has also taught me that the future of game dev is expanding.   Even though DirectX is common and popular, there are hundreds of other programming languages out there, all that can be used for game development, and having knowledge of more than one can help increase your chances of getting a career in this line of work. Just as how creativity is not restricted to one idea, game dev is not just restricted to DirectX.

REFERENCES

- "2D Collision Detection". *Mozilla Developer Network*. 14 April 2015. https://developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection. Web.
- "CSS Image Opacity/Transparency". *w3schools.com.* 14 April 2015. http://www.w3schools.com/css/css_image_transparency.asp. Web.
- "HTML5 Audio". *w3schools.com.* 14 April 2015. http://www.w3schools.com/html/html5_audio.asp. Web.
- "HTML Canvas". *w3schools.com*. 10 April 2015. http://www.w3schools.com/html/html5_canvas.asp. Web.
- "HTML canvas beginPath() method". *w3schools.com*. 11 April 2015. http://www.w3schools.com/tags/canvas_beginpath.asp. Web.
- "HTML Canvas Drawing". *w3schools.com.* 10 April 2015. http://www.w3schools.com/canvas/canvas_drawing.asp. Web.
- "HTML5 canvas fillRect() method". *w3schools.com.* 10 April 2015. http://www.w3schools.com/tags/canvas_fillrect.asp. Web.
- "JavaScript HTML DOM EventListener". *w3schools.com.* 11 April 2015. http://www.w3schools.com/js/js_htmldom_eventlistener.asp. Web.
- "JavaScript Tutorials". *w3schools.com*. 9 April 2015. http://w3schools.com/js/default.asp. Web.
- "KeyboardEvent keyCode Property". *w3schools.com.* 11 April 2015. http://www.w3schools.com/jsref/event_key_keycode.asp. Web.
- "The History of DirectX". *CodingUnit*. http://www.codingunit.com/the-history-of-directx. 8 April 2015. Web.

- Brockschmidt, Kraig. "My Take on HTML/JS vs. C#/XAML vs. C++/DirectX (choosing a language for a Windows Store app)". *kraig brockschmidt*. 17 Jan 2013. 9 April 2015. http://www.kraigbrockschmidt.com/2013/01/17/html-javascript-xaml-directx-language-windows-store-app/. Web Blog.
- Rosen, David. "Why you should use OpenGL and not DirectX". *Wolfire Blog Games*. 8 Jan 2010. 8 April 2015. http://blog.wolfire.com/2010/10/Why-you-should-use-OpenGL-and-not-DirectX. Web Blog.

APPENDIX