GD 0
Basic Shapes

Early game development required knowledge of computer architecture. Once this knowledge was acquired, the programmer had to write code to control the flow of data between different units within the computer. This style of development was error prone and time consuming.

Modern game development is **composed of two levels**

> **1. updating game objects**
> **2. drawing (or rendering) game objects**.

The programmer is only responsible for write code to update the game object. The tool use, called **HTML 5 Canvas**, will draw the game objects for us. This simplifies game development into updating game objects and then giving input commands to the tool to render the game object on the screen.

1. Basic shapes 2. Canvas tool to draw shape 3. Using variables to describe a shape

0. INTRODUCTION

Early game development was focused on hardware control. In order to generate an image onto the screen, the programmer had to understand the computer architecture. Once the computer architecture was understood, the programmer had the knowledge to understand how to control the flow of data between internal units of the computer. This was important because the programmer now knows how to load, store, and then change data within the computer.

Having this knowledge was important for displaying the game onto a screen and then updating the game objects themselves (positioning, vitality, and etc).

Display: The programmer had to manually tell the computer to load an image into a frame buffer. Once an image was in a **frame buffer**, it is displayed onto the screen, like a CRT or computer monitor. If the game had multiple game objects, the programmer had to write code to load all of those game objects into the frame buffer.

Updating game objects: The programmer also had the responsibility of writing code to update the game objects. For example, if the programmer was creating Space Invaders, then the programmer **was also responsible** for writing the code that makes the enemy space ships fall from the sky. This also meant writing code to control adders within the computer so that the enemy space ships move left and right while falling down.

Early game development involved hardware programming and knowing how data flows from one internal unit of computation to another unity of computation. This manual control of hardware was difficult and error prone. **The programmer had to do everything!!!**

Modern Game Development: Game development has advanced to the point where the programmer no longer needs to have knowledge of the computer architecture of the computer that will host the game.

Instead, game development now uses tools that do the work on behalf of the programmer. The programmer simply uses the available tools and focuses on updating game objects rather displaying the game objects ➔ we give input commands to the tool and the tool takes our commands and renders the game objects on the screen on our behalf.

Our work uses predefined function calls (ie. premade actions) using **HTML 5 Canvas**. Using Canvas, we can now see game development as to different levels.

1. **The first level** is writing code to **update game objects** (x, y position, vitality, color, and etc). **We stay at this level exclusively!!!**

2. **The second level** is the **actual rendering (or drawing**) of the game object based on our input commands. **We don't do this anymore**. **Instead, the Canvas tools will do the rendering.**

1. HTML5 Canvas

Game development is art!!! Every game is composed of basic shapes and these basic shapes become the game objects. The tool we will use to draw the game objects is called **HTML 5 Canvas**. Remember the paragraph tag <p> </p>? It is an element of a website and is used as a placeholder of data.

Source: fineartamerica.com

Similarly, we also have the `<canvas> </canvas>` element and the canvas element is an area where we draw our game objects. Once we link up JS with HTML 5 canvas using `document.getElementById()`, we are given a tool set that is used to draw our game objects.

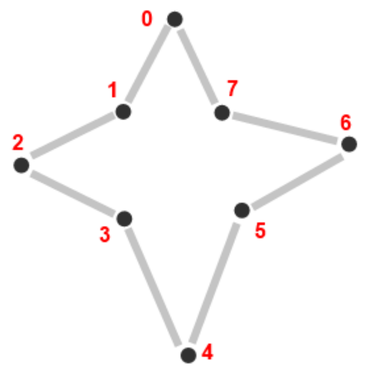
The tool set is actually a large collection of premade function calls. All we have to do is use them by calling the function name.

1.1. What is a shape?

It is an array of points. When we connect these points together, the points create an outline of the shape that we see.

Canvas starts by asking us to pick an (x,y) point as the starting point and then we pick more points to following the starting point and so on.

So, we can think of canvas as a literal painting canvas and the predefined functions allow us to pick up our pen and place them down onto the "**canvas**" to mark the next (x,y) point. We then connect the points together to create a shape.



When we connect all of the points together with a line, it creates a shape. **Remember that the first point, point 0, and last point, point 7, must connect with each other.** If they do no connect with each other, the line never ends and we don't have a shape.

In HTML 5 Canvas, we begin drawing by typing **ctx.beginPath()** and this allows us to start with **point 0**. We tell the website to **end and connect all points** with a line by typing **ctx.fill().**

2. HTML 5 CANVAS GAME DEV – 2 LEVELS

Level 1 Input Commands – declaring variables and loading data into them. This is a sample of the code that we the programmer will write. **We simply give input commands to the tool → input commands go in between the open and closing parenthesis of a function call.**

```
var canvas = document.getElementById('gameEnv');
var ctx    = canvas.getContext('2d');

 // -- vars below are input commands
var
xPos    = 20,
yPos    = 20,
rWidth  = 100,
rHeight = 100;
```

Level 2 Draw – **calling predefined functions. The input commands that were created are then given to tool and tool draws the rectangle for us**

The code below will draw the rectangle for us. We don't have to write code to draw it manually.

Instead, the code in **blue** is the rectangle function and it draws the rectangle for us. Notice that we give our input commands of x and y position, width, and height of our rectangle.

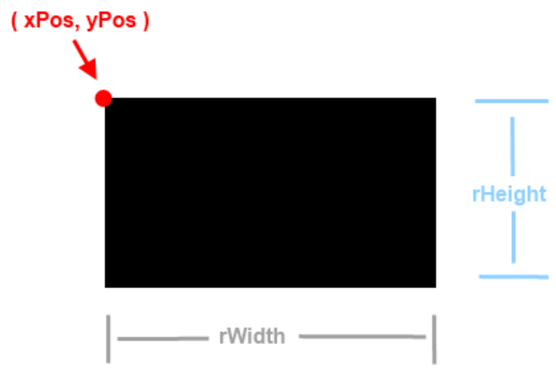
Finally, the code in **green** below will connect the dots to create the shape.

```
ctx.rect(xPos, yPos, rWidth, rHeight);
ctx.fill();
```

2.1. BRING IT ALL TOGETHER

We created 4 variables and they are called `xPos, yPos, rWidth, and rHeight.`

The variables `xPos and yPos` are the **starting point** of the rectangle. The variable `rWidth` tells us how **wide** we want our rectangle to be. The variable `rHeight` tells us how **tall** we want our rectangle to be. These become the input commands to the tool, which is `ctx.rect()` .



Finally, we use `ctx.rect(xPos, yPos, rWidth, rHeight);` to indicate that the **tool should draw a rectangle.**

The code `ctx.fill();` fills in the rectangle with a color. If we don't fill in the rectangle, we don't see it!!!

NOTE: we don't draw the rectangle because the tool does that work for us. We simply tell the tool the starting point of the rectangle, which is `xPos, yPos`. Next, we tell the tool that the rectangle's width is `rWidth` and the height is `rHeight`. The tool will then draw for us.

3. Write the code below in between <script> </script>. The **large, green banner** is your landmark. **Go to the coding website and look for it.** Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

3.1. JS Challenge 1 – Write the input commands for paddle 2. Look for

```
/* ===================================
*  Input command ( vars ) for paddle 2
*
* ==================================== */
```

and put the input commands below that line.

1. Create the variable xPos2 and load the data 500 into it
2. Create the variable yPos2 and load the data 200 into it
3. Create the variable rWidth2 and load the data 50 into it
4. Create the variable rHeight2 and load the data 300 into it

3.2. JS Challenge 2 – Draw paddle 2. Look for

```
/* ===================================
* Code to draw 2nd paddle
*
* ==================================== */
```

1. Give the input commands to `ctx.rect()`
2. Use `ctx.fill();` to actually connect all points with a line to complete the shape

When done, press the "**Run**" button and you should see a second, gold rectangle. Change the color of this second rectangle if you like.

4. PRIMITIVE SHAPES

The code above uses `ctx.rect()`, which is part of the HTML 5 Canvas tool set, to create a primitive shape of the rectangle.

What are primitive shapes? These are basic shapes that are **used so often** that HTML 5 Canvas converted them into premade objects. Other primitive shapes are square and circle.

4.1. CIRCLE COMMAND – notice it is "arc"

arc(x, y, radius, startAngle, endAngle, ccw)

Draws an arc which is centered at *(x, y)* position with radius ***radius*** starting at ***startAngle*** and ending at ***endAngle*** going in the given direction indicated by ***ccw*** (defaulting to clockwise).

5. Write the code below in between <script> </script>. The large, green banner is your landmark. Go to the coding website and look for it. Next, write the code below underneath the large, green banner. Write all of it, color code is for explanation.

```
/* ====================================
* Input command ( vars ) of circle
*
* ==================================== */
var
cXPos   = 275,
cYPos   = 230,
cRad    = 30,               // -- radius
cSAngle = 0,                // -- circle start angle
cEAngle = Math.PI * 2,      // -- circle end angle
ccw     = true;             // -- counter clock wise = true
```

Explanation

We are **declaring variables and then loading data into them**. These become input commands and they **describe** how the circle will look once it is drawn.

6. Write the code below in between <script> </script>. The **large, green banner** is your landmark. **Go to the coding website and look for it.** Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
/* ===================================
* Code to draw circle
*
* =================================== */
ctx.fillStyle = "#006500";
ctx.beginPath();
ctx.arc( cXPos, cYPos, cRad, cSAngle, cEAngle, ccw );
ctx.fill();
```

Explanation

We use the variables that were just created in the previous section as input commands to tell the tool to draw a circle. In this case, the `ctx.arc()` tells the tool to **draw** a circle.

The code in **blue** tells the website that we want a new starting point, which is point 0. The code in **grey** tells the website that we have finished and want to connect all points together.

You can change the `ctx.fillStyle` to be the color that you like. Remember to put double quotation, and hash tag.

CONTINUE TO THE NEXT PAGE

7. ANIMATION USING CANVAS

Right now, the game objects are static and don't change. They are basically frozen in time. Game objects at least have to change position (either x or y position) in order to create animation.

How do we make our game characters move in order create animation? We need to make the game object dynamic by changing their positioning and **then redrawing** the game objects in their new, and updated x,y position.

7.1. WIPE IT OFF, ALL OF IT!!!

Here is the unique part about game development. Everything you see on the computer screen or monitor must be ***refreshed*** every 60 Hz. What does it mean to ***refresh*** something?

Refresh →It means that the computer screen must be cleared of its content. This means that ***every*** game object ***must be redrawn onto the computer screen every 16 milliseconds***.

7.2. `window.requestAnimationFrame(draw_game);`

How does the computer know when to refresh the screen? There is a special function called `window.requestAnimationFrame( functionName );` This function will determine when it is time to start the next refresh cycle.

When it is time to refresh the screen, the function name that will draw all of the game object is called. In this case, the function that does the drawing is put inside the open and closing parenthesis → functionName.

Tricky → **the function that does the drawing then calls itself over and over again**. This is how the game is able to continuously clear the screen and then redraw the game objects in their updated positions.

7.2. Basic Steps

3. Clear the canvas to clear the screen
4. Update the x and y position of a game object
5. Give the input commands to the tool – level 1
6. Call the predefined functions to draw the shape – level 2
7. Wait for next refresh cycle
8. Go back to step 1 and do it again

8. Write the code below in between <script> </script>. The **large, green banner** is your landmark. **Go to the coding website and look for it.** Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
      /* ===================================
      * Redraw every 16 milliseconds
      *
      * =================================== */
     redrawCon = window.requestAnimationFrame(draw_game);
```

Explanation

The important part is `window.requestAnimationFrame(draw_game);` This premade function definition has a timer that is already configured so we don't have to do anything. The only input we give is the name of the function definition that will redraw all of the game objects in their updated positions. In this case, the function definition is named "`draw_game`" and it called every 16 milliseconds.

9. Write the code below in between <script> </script>. The **large, green banner** is your landmark. **Go to the coding website and look for it.** Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
/* ===================================
*  CAll itself in a loop
*
* ==================================== */
function draw_game()
{
      // -- wipe it all off
      ctx.clearRect(0, 0, canvas.width, canvas.height);
      sampleCount++;
      console.log( "sampleCount: " + sampleCount );

      // -- draw rectangle
      xPos += 3;
      yPos += 3;
      ctx.fillStyle = "#ffd700";
      ctx.beginPath();
      ctx.rect(xPos, yPos, rWidth, rHeight);
      ctx.fill();

      // -- draw rectangle 2
      xPos2 += 3;
      yPos2 += 3;

      ctx.fillStyle = "#606060";
      ctx.beginPath();
      ctx.rect(xPos2, yPos2, rWidth2, rHeight2);
      ctx.fill();
```

```
        // -- draw circle
        cXPos += 3;
        cYPos -= 3;

        ctx.fillStyle = "#a56500";
        ctx.beginPath();
        ctx.arc( cXPos, cYPos, cRad, cSAngle, cEAngle, ccw );
        ctx.fill();

        // -- prevent infinite loop
        if( sampleCount >= 400 )
        {
          window.cancelAnimationFrame(redrawCon)
        }
        else
        {
          redrawCon = window.requestAnimationFrame(draw_game);
        }

}
```

Explanation

Every 16 milliseconds, we have to redraw every game object. The function definition above does that by first clearing the screen. The code

```
ctx.clearRect(0, 0, canvas.width, canvas.height);
```

clears the entire canvas. Once the entire canvas is cleared, the screen is completely blank and we start to redraw again.

Notice that we **add 3** to the x and y position of **all of our game objects**. By doing so, we just wrote the code that updates each game object and **we just completed level 1**. In the code above, each game object moves 3 steps in the downward, right direction.

After we updated each game object, **we focus on level 2**, which is using the tool to redraw each game object in their new, and updated, (x, y) position. **The tool is composed of premade function definitions**. We simply give input commands to each function definition and this tells the tool to redraw

```
ctx.rect(xPos, yPos, rWidth, rHeight);

ctx.rect(xPos2, yPos2, rWidth2, rHeight2);

ctx.arc( cXPos, cYPos, cRad, cSAngle, cEAngle, ccw );
```

Since we updated each game object's x and y position, **each game object will be redrawn in a different position** and this is how dynamic game play is created.