Computer Science Principles

Web Programming

JavaScript Canvas Programming

CHAPTER 2: GAME DESIGN

DR. ERIC CHOU

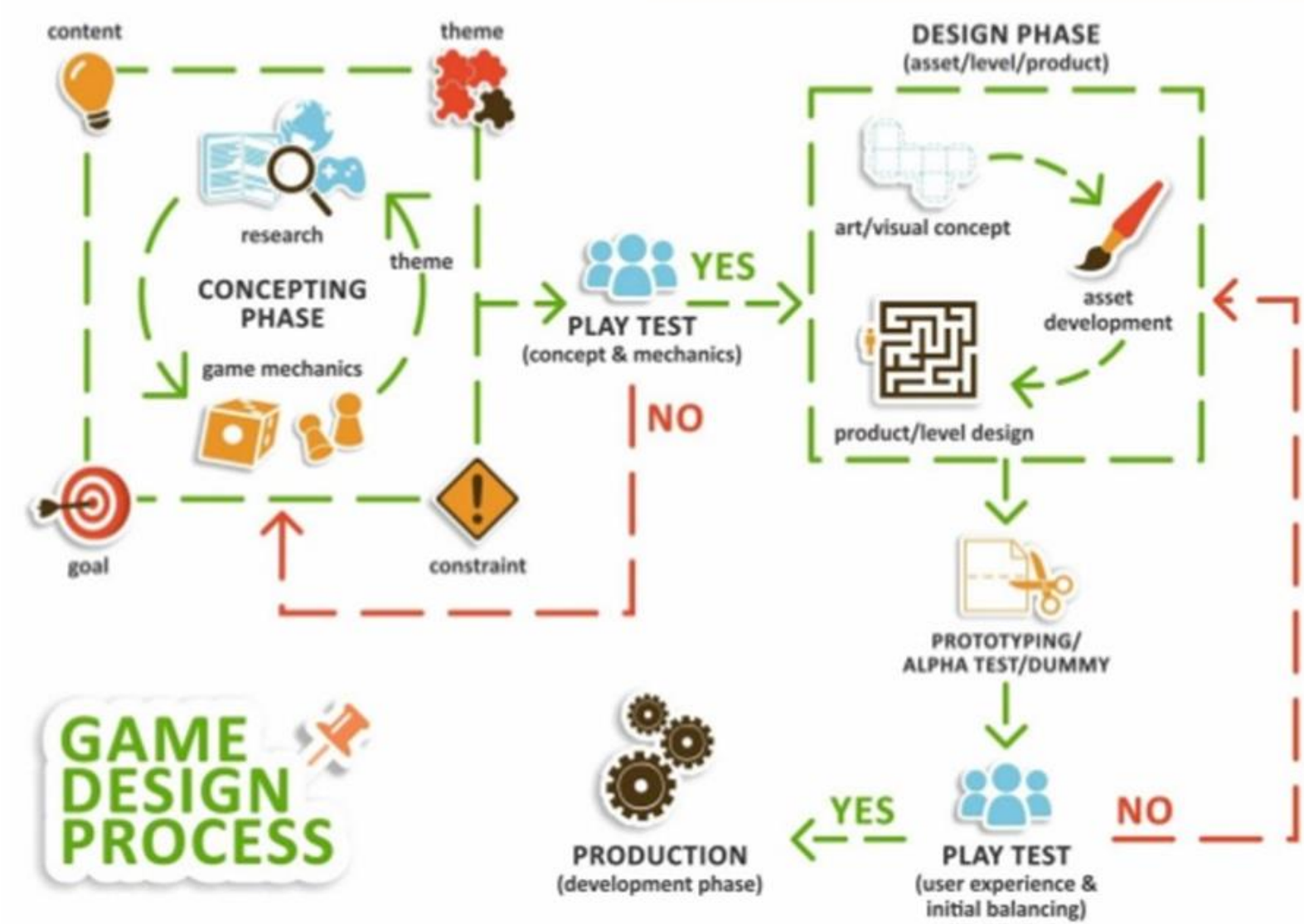
IEEE SENIOR MEMBER

Overview

- Game Design Overview (flight game)
- Canvas Container
- JavaScript Object-Oriented Paradigm
- Game Components
- Model-View-Controller Design
  - Scene Design
  - Scoring and Winning Condition
  - Art Design

Overview

LECTURE 1

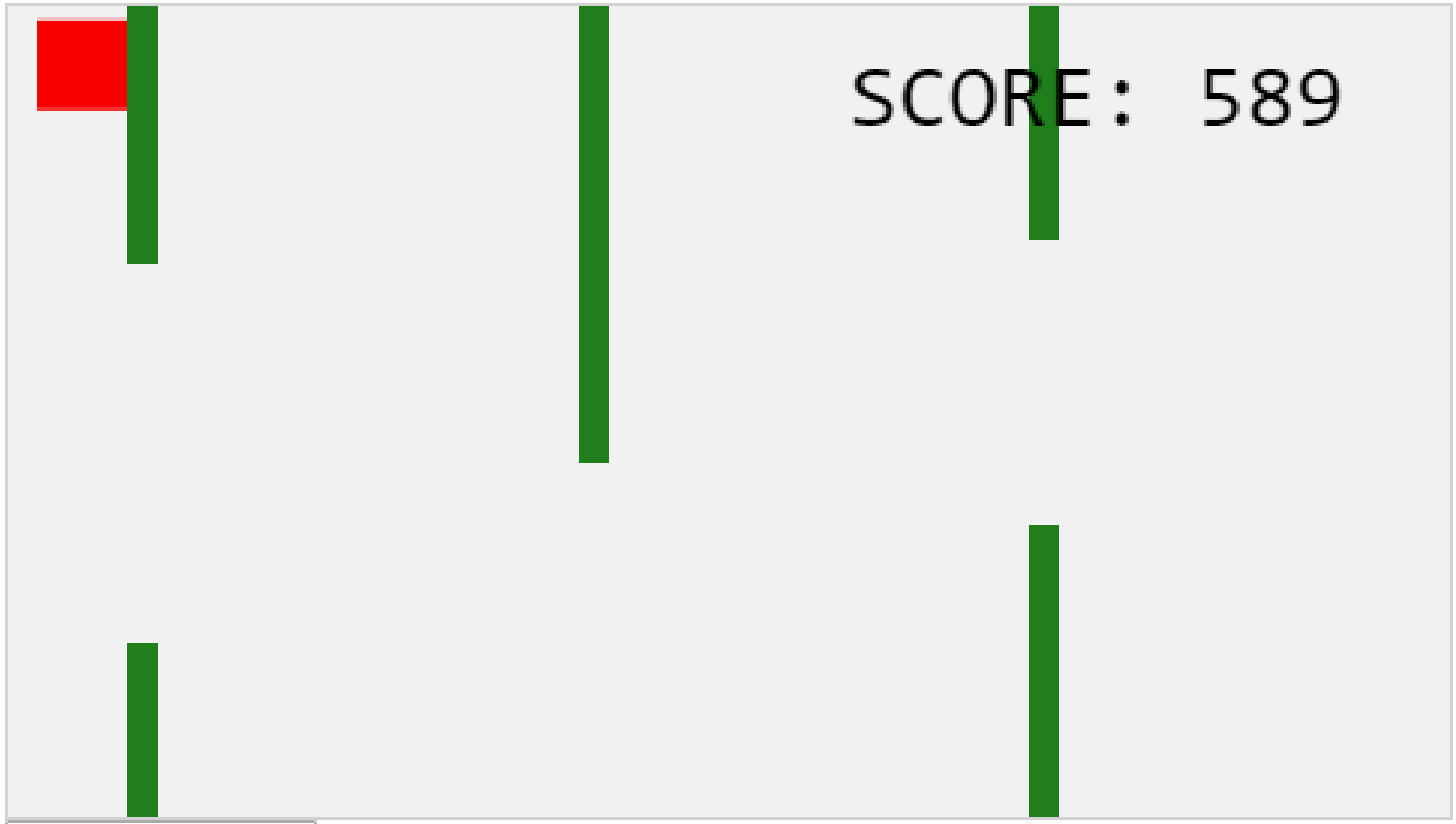

content
theme
research
theme
CONCEPTING PHASE
game mechanics
goal
constraint
YES
PLAY TEST
(concept & mechanics)
NO
DESIGN PHASE
(asset/level/product)
art/visual concept
asset development
product/level design
PROTOTYPING/
ALPHA TEST/DUMMY
PLAY TEST
(user experience & initial balancing)
YES
NO
PRODUCTION
(development phase)
GAME DESIGN PROCESS

Canvas 2D Flight Game

- In this step-by-step tutorial we create a simple flight game written entirely in pure **JavaScript** and rendered on **HTML5** <**canvas**>.
- Every step has editable, live samples available to play with so you can see what the intermediate stages should look like. You will learn the basics of using the <**canvas**> element to implement fundamental game mechanics like rendering and moving images, collision detection, control mechanisms, and winning and losing states.

Canvas 2D Flight Game

- To get the most out of this series of articles you should already have basic to intermediate JavaScript knowledge. After working through this tutorial you should be able to build your own simple Web games.



ACCELERATE

Use the ACCELERATE button to stay in the air

How long can you stay alive?

Game Canvas

LECTURE 2

HTML Canvas

- The <**canvas**> element is perfect for making games in HTML.
- The <**canvas**> element offers all the functionality you need for making games.
- Use JavaScript to draw, write, insert images, and more, onto the <**canvas**>.

.getContext("2d")

- The <canvas> element has a built-in object, called the getContext("2d") object, with methods and properties for drawing.
- You can learn more about the <canvas> element, and the getContext("2d") object.

Dynamic Canvas Versus Static Canvas

- So far, we have been working on only static canvas which is assigned at a fixed location just like other HTML elements.
- In this chapter, we will be using a dynamic canvas approach by using the **createElement**("**id**") method

Dynamic Page

- Easier for programming point of view: reset, re-draw, re-shape, and hiding/display.
- Better interface with other program parts.
- Create and manage canvas completely using JavaScript.
- Multiple canvas areas
- Longer code

dynamic.html

```
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
<style>
canvas {
    border: 5px solid #d3d3d3;
    background-color: #f1f1f1;
}
</style>
</head>
<body onload="startGame()">
<script>
function startGame() {
    myGameArea.start();
}
var myGameArea = {
    canvas : document.createElement("canvas"),
    start : function() {
        this.canvas.width = 480;
        this.canvas.height = 270;
        this.context = this.canvas.getContext("2d");
        document.body.insertBefore(this.canvas,
        document.body.childNodes[0]);
    }
}
</script>
<p>We have created a dynamic game area!</p>
</body>
</html>
```

Dynamic Page

- The object myGameArea will have more properties and methods later.
- The function startGame() invokes the method start() of the myGameArea object.
- The start() method creates a <canvas> element and inserts it as the first childnode of the <body> element.

We have created a dynamic game area!

Static Page

- <canvas> is an element.

static.html

```
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
<style>
canvas {
    border: 5px solid #d3d3d3;
    background-color: #f1f1f1;
}
</style>
</head>
<body onload="startGame()">
<canvas id="canvas" width="480" height="270"></canvas>
<script>
</script>
<p>We have created a static game area!</p>
</body>
</html>
```

Game Components

LECTURE 3

Game Area and Game Components

Add a Component

- Make a component constructor, which lets you add components onto the **gamearea**.
- The object constructor is called component, and we make our first component, called **myGamePiece**:

```
var myGamePiece;

function startGame() {
  myGameArea.start();
  myGamePiece = new component(30, 30, "red", 10, 120);
}

function component(width, height, color, x, y) {
  this.width = width;
  this.height = height;
  this.x = x;
  this.y = y;
  ctx = myGameArea.context;
  ctx.fillStyle = color;
  ctx.fillRect(this.x, this.y, this.width, this.height);
}
```

Frames

- To make the game ready for action, we will update the display 50 times per second, which is much like frames in a movie.
- First, create a new function called **updateGameArea()**.
- In the **myGameArea** object, add an interval which will run the **updateGameArea()** function every 20th millisecond (50 times per second). Also add a function called **clear()**, that clears the entire canvas.
- In the **component** constructor, add a function called **update()**, to handle the drawing of the component.
- The **updateGameArea()** function calls the **clear()** and the **update()** method.

```
var myGameArea = {
  canvas : document.createElement("canvas"),
  start : function() {
    this.canvas.width = 480;
    this.canvas.height = 270;
    this.context = this.canvas.getContext("2d");
    document.body.insertBefore(this.canvas, document.body.childNodes[0]);
    this.interval = setInterval(updateGameArea, 20);
  },
  clear : function() {
    this.context.clearRect(0, 0, this.canvas.width, this.canvas.height);
  }
}
```

```
function component(width, height, color, x, y) {
  this.width = width;
  this.height = height;
  this.x = x;
  this.y = y;
  this.update = function(){
    ctx = myGameArea.context;
    ctx.fillStyle = color;
    ctx.fillRect(this.x, this.y, this.width, this.height);
  }
}

function updateGameArea() {
  myGameArea.clear();
  myGamePiece.update();
}
```

Make it Move

move.html

- To prove that the red square is being drawn 50 times per second, we will change the x position (horizontal) by one pixel every time we update the game area:

```
function updateGameArea() {
    myGameArea.clear();
    myGamePiece.x += 1;
    myGamePiece.update();
}
```

Clear

Update

Redraw

Why Clear The Game Area?

moveNoClear.html

- It might seem unnecessary to clear the game area at every update. However, if we leave out the clear() method, all movements of the component will leave a trail of where it was positioned in the last frame:

```
function updateGameArea() {
  // myGameArea.clear();
  myGamePiece.x += 1;
  myGamePiece.update();
}
```

Change the Size

changeSize.html

- You can control the width and height of the component:
- **Example**
  - Create a 10x140 pixels rectangle:

```
function startGame() {
    myGameArea.start();
    myGamePiece
        = new component(140, 10, "red", 10, 120);
}
```

Change the Color

changeColor.html

- You can control the color of the component:
- **Example**

```
function startGame() {
  myGameArea.start();
  myGamePiece
= new component(30, 30, "blue", 10, 120);
}
```

Change the Color

changeColor2.html

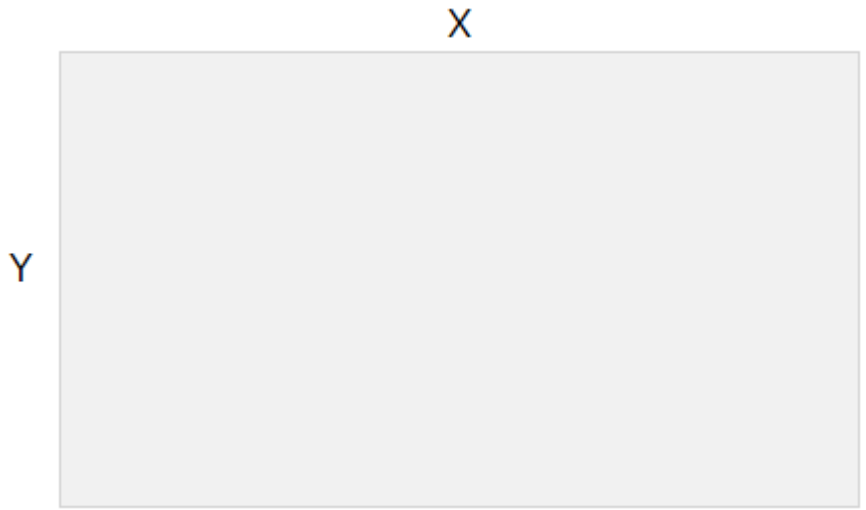
- You can also use other colorvalues like hex, rgb, or rgba:
- **Example**

```
function startGame() {
  myGameArea.start();
  myGamePiece = new component(30, 30, "rgba(0, 0,
255, 0.5)", 10, 120);
}
```

Change the Position

changePosition.html

- We use x- and y-coordinates to position components onto the game area.
- The upper-left corner of the canvas has the coordinates (0,0)
- Mouse over the game area below to see its x and y coordinates:



Change the Position

changePosition.html

- You can position the components wherever you like on the game area:
- **Example:**

```
function startGame() {
  myGameArea.start();
  myGamePiece = new component(30, 30, "red", 2, 2);
}
```

Multiple Components

mc.html

- You can put as many components as you like on the game area:
- **Example:**

```
var redGamePiece, blueGamePiece, yellowGamePiece;
function startGame() {
  redGamePiece = new component(75, 75, "red", 10, 10);
  yellowGamePiece = new component(75, 75, "yellow", 50, 60);
  blueGamePiece = new component(75, 75, "blue", 10, 110);
  myGameArea.start();
}
function updateGameArea() {
  myGameArea.clear();
  redGamePiece.update();
  yellowGamePiece.update();
  blueGamePiece.update();
}
```

Multiple Components

mc.html

- You can put as many components as you like on the game area:
- **Example:**

```
var redGamePiece, blueGamePiece, yellowGamePiece;
function startGame() {
  redGamePiece = new component(75, 75, "red", 10, 10);
  yellowGamePiece = new component(75, 75, "yellow", 50, 60);
  blueGamePiece = new component(75, 75, "blue", 10, 110);
  myGameArea.start();
}
function updateGameArea() {
  myGameArea.clear();
  redGamePiece.update();
  yellowGamePiece.update();
  blueGamePiece.update();
}
```

Multiple Components

mc2.html

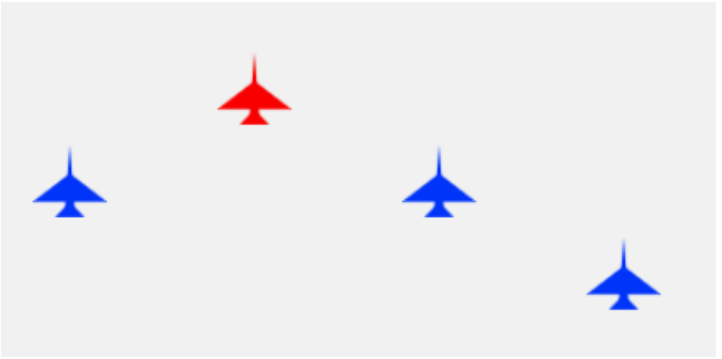
- Make all three components move in different directions:
- **Example:**

```
function updateGameArea() {
  myGameArea.clear();
  redGamePiece.x += 1;
  yellowGamePiece.x += 1;
  yellowGamePiece.y += 1;
  blueGamePiece.x += 1;
  blueGamePiece.y -= 1;
  redGamePiece.update();
  yellowGamePiece.update();
  blueGamePiece.update();
}
```

Fighters
fighter.html

MULTIPLE FIGHTERS IN A SQUADRON:

```
function component(width, height, color, x, y) {
    this.width = width;
    this.height = height;
    this.x = x;
    this.y = y;
    this.update = function(){
        let xx = this.x;
        let yy = this.y;
        let h = this.height;
        let w = this.width;
        let w2 = this.width/2;
        let h2 = this.height/2;
        let h3 = this.height/3;
        let h32 = this.height*2/3;
        ctx = myGameArea.context;
        ctx.beginPath();
        ctx.moveTo(xx+w2, yy);
        ctx.lineTo(xx+w2-3, yy+h-1);
        ctx.lineTo(xx+w2+3, yy+h-1);
        ctx.lineTo(xx+w2, yy);
        ctx.closePath();
        ctx.fillStyle = color;
        ctx.lineWidth = "4";
        ctx.fill();
        ctx.beginPath();
        ctx.moveTo(xx+w2, yy+h3+2);
        ctx.lineTo(xx, yy+h32+4);
        ctx.lineTo(xx+w, yy+h32+4);
        ctx.lineTo(xx+w2, yy+h3+2);
        ctx.closePath();
        ctx.fillStyle = color;
        ctx.lineWidth = "1";
        ctx.fill();
        ctx.beginPath();
        ctx.moveTo(xx+w2, yy+h32+4);
        ctx.lineTo(xx+w2-8, yy+h-1);
        ctx.lineTo(xx+w2+8, yy+h-1);
        ctx.lineTo(xx+w2, yy+h32+4);
        ctx.closePath();
        ctx.fillStyle = color;
        ctx.lineWidth = "1";
        ctx.fill();
    }
}
```

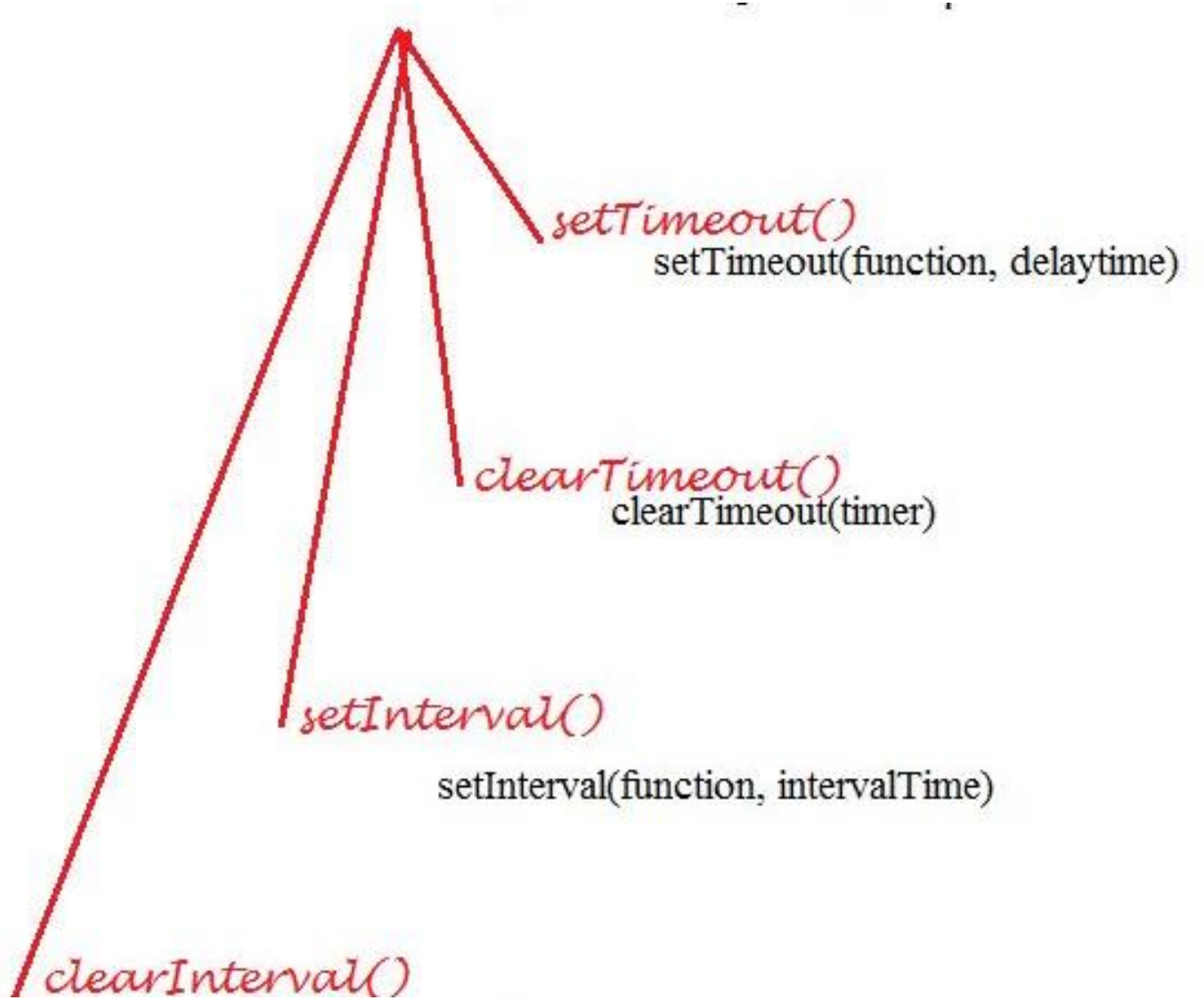


Fighters

fighters2.html

- Reset y to loop back
- Fighters moving in different directions (vectors):

```
var d = [[0, -1], [-0.1, -0.9],
         [-0.3, -0.7], [0.1, -0.9],
         [0.3, -0.7], [-0.5, -0.5], [0.5, -0.5]
        ];  // all moving forward
```



Timer Methods

Synchronous updates:

```
timer = setInteval(
            update_function,
            intervalTime)
clearInterval(timer)
```

Asynchronous updates:

```
timer = setTimeout(
            update_function,
            delayTime)
clearTimeout(timer)
```

Basic Animation

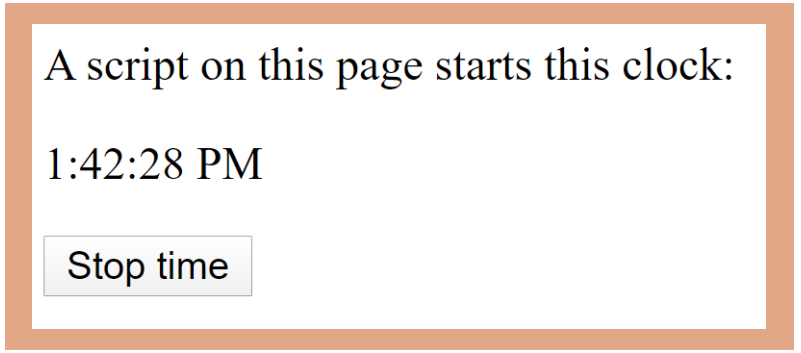
- Periodic Update (Update -> Redraw -> Hold Screen)

```
while (!clear interval){
      sleep(intervalTime);
      myTimer(); // update for data model and
                              // redraw the screen
}
```

Window clearInterval() Method

Example

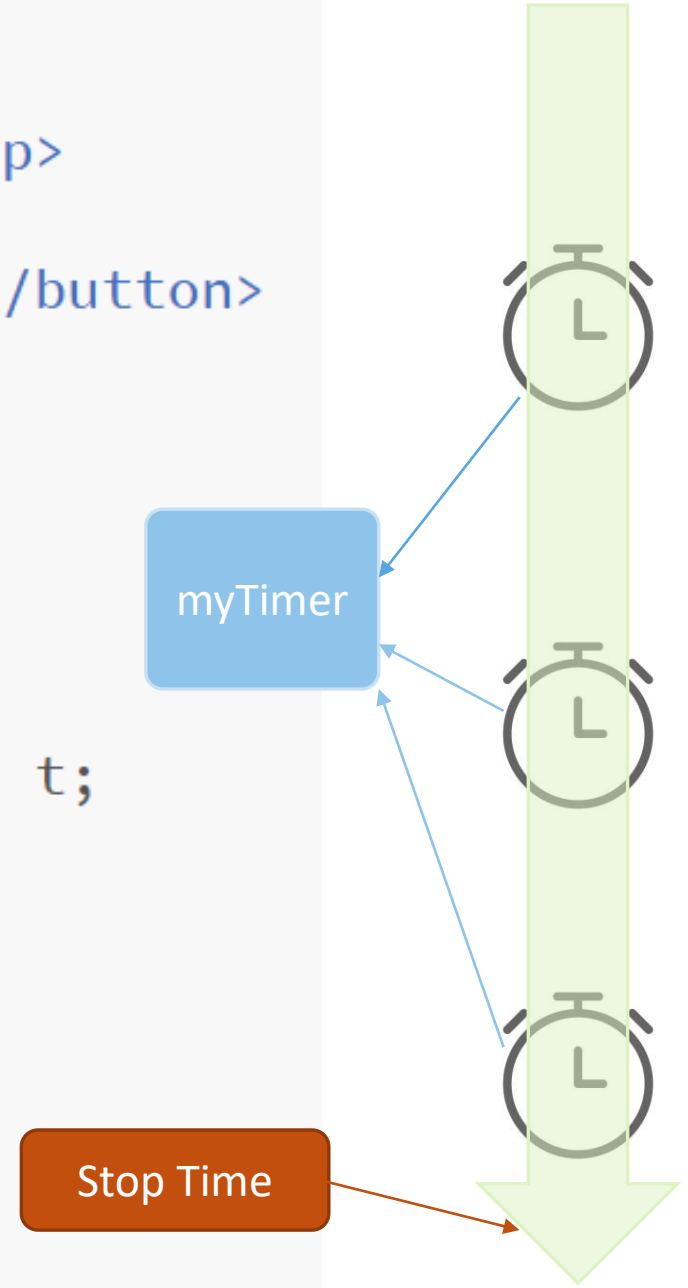
- Display the current time (the setInterval() method will execute the "myTimer" function once every 1 second).
- Use clearInterval() to stop the time:




```
var myVar = setInterval(myTimer, 1000);
function myTimer() {
  var d = new Date();
  var t = d.toLocaleTimeString();
  document.getElementById("demo").innerHTML = t;
}
function myStopFunction() {
  clearInterval(myVar);
}
```

```html
<html>
<body>
<p>A script on this page starts this clock:</p>
<p id="demo"></p>
<button onclick="myStopFunction()">Stop time</button>

<script>
var myVar = setInterval(myTimer, 1000);
function myTimer() {
  var d = new Date();
  var t = d.toLocaleTimeString();
  document.getElementById("demo").innerHTML = t;
}
function myStopFunction() {
  clearInterval(myVar);
}
</script>
</body>
</html>
```



Game Controllers and Events

LECTURE 4

Inputs (Events)

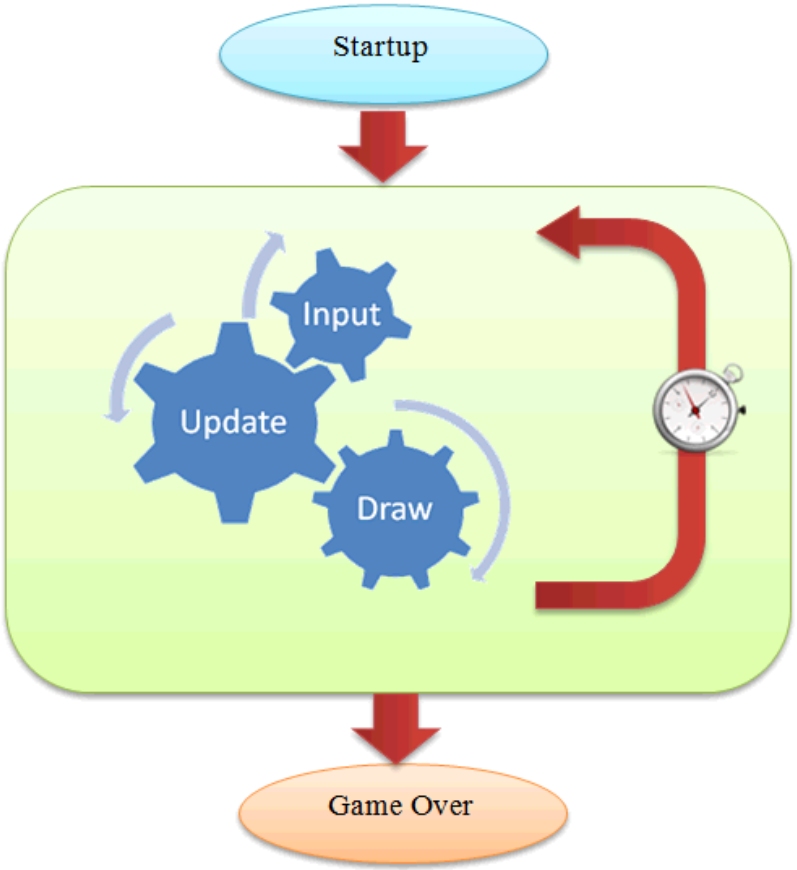
- Button
- Keyboard
- Mouse
- Touch Screen

Game Loop

- Uniform Game Loop:

```
this.interval =
    setInterval(updateGameArea, 200);
    // Every 200 milli-secs.
```

- Update of Data Model and repaint of canvas.



Update Model:

```
this.newPos = function() {
    this.x += this.speedX;
    this.y += this.speedY;
}
```

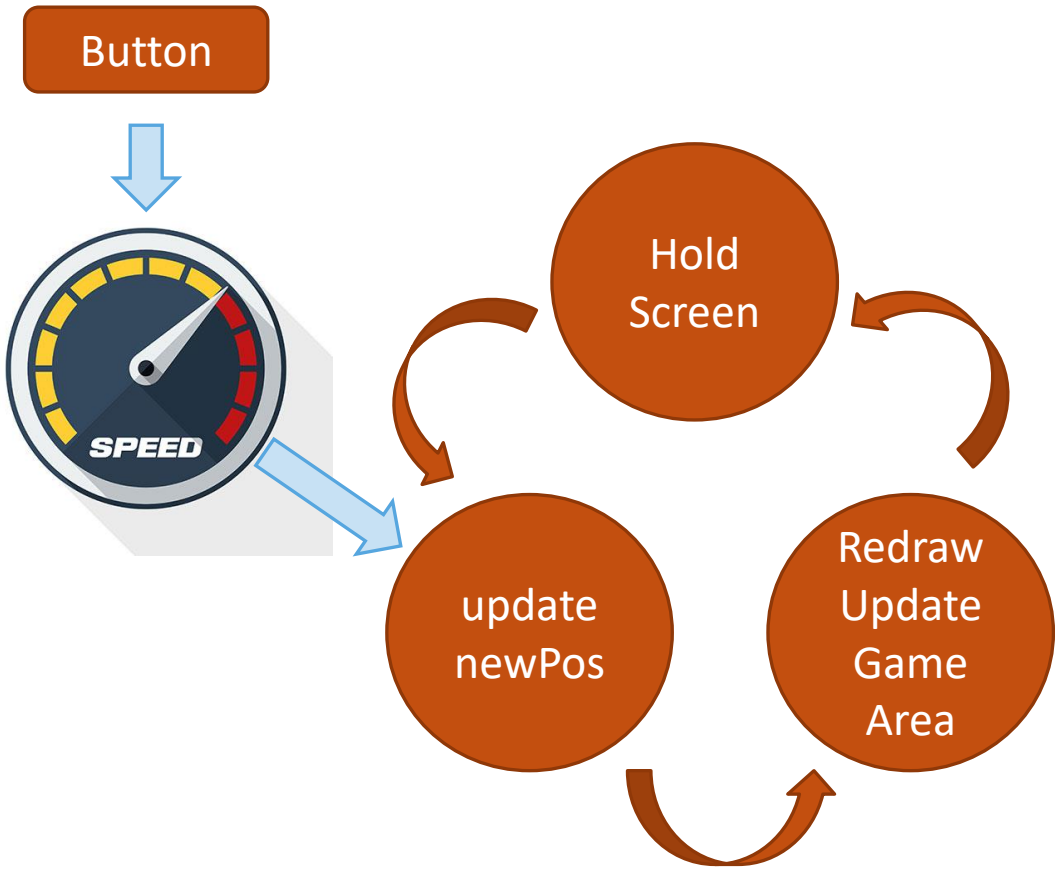
Redraw:

```
function updateGameArea() {
    myGameArea.clear();
    blueFighter.newPos();
    blueFighter.update();
}
```

Game Loop

Input (change speed)

```
96  function moveup() {
97      blueFighter.speedY -= 1;
98  }
99
100 function movedown() {
101     blueFighter.speedY += 1;
102 }
103
104 function moveleft() {
105     blueFighter.speedX -= 1;
106 }
107
108 function moveright() {
109     blueFighter.speedX += 1;
110 }
111 </script>
112 <div style="text-align:center;width:480px;">
113   <button onclick="moveup()">UP</button><br><br>
114   <button onclick="moveleft()">LEFT</button>
115   <button onclick="moveright()">RIGHT</button><br><br>
116   <button onclick="movedown()">DOWN</button>
117 </div>
```



Accelerated Fighter:

MoveByButton.html

- Up: speed up toward north or slow down toward south.
- Down: speed up toward south or slow down toward north.
- Left: speed up toward left or slow down toward right.
- Right: speed up toward right or slow down toward south.

Cruising Fighter:

MoveByButton2.html

- Up: cruise up at same speed.
- Down: cruise down at same speed.
- Left: cruise left at same speed.
- Right: cruise right at same speed.

Stop Moving:

MoveByButton3.html

- If you want, you can make the red square stop when you release a button.
- Add a function that will set the speed indicators to 0.
- To deal with both normal screens and touch screens, we will add code for both devices:

- Example:

```
function stopMove() {
  myGamePiece.speedX = 0;
  myGamePiece.speedY = 0;
}
</script>
<button onmousedown="moveup()" onmouseup="stopMove()"
        ontouchstart="moveup()">UP</button>
<button onmousedown="movedown()" onmouseup="stopMove()"
        ontouchstart="movedown()">DOWN</button>
<button onmousedown="moveleft()" onmouseup="stopMove()"
        ontouchstart="moveleft()">LEFT</button>
<button onmousedown="moveright()" onmouseup="stopMove()"
        ontouchstart="moveright()">RIGHT</button>
```

Accelerated Fighter Controlled by Keyboard

MoveByButton4.html

- Event Source
- Event Object
- Event Listener
- Event Handler

Keyboard as Controller

- We can also control the red square by using the arrow keys on the keyboard.
- Create a method that checks if a key is pressed, and set the key property of the myGameArea object to the key code. When the key is released, set the key property to false:
- **Example:**

```
var myGameArea = {
  canvas : document.createElement("canvas"),
  start : function() {
    this.canvas.width = 480;
    this.canvas.height = 270;
    this.context = this.canvas.getContext("2d");
    document.body.insertBefore(this.canvas, document.body.childNodes[0]);
    this.interval = setInterval(updateGameArea, 20);
    window.addEventListener('keydown', function (e) {
      myGameArea.key = e.keyCode;
    })
    window.addEventListener('keyup', function (e) {
      myGameArea.key = false;
    })
  },
  clear : function(){
    this.context.clearRect(0, 0, this.canvas.width, this.canvas.height);
  }
}
```

Multiple Keys Pressed

MoveByButton5.html

- What if more than one key is pressed at the same time?
- In the example above, the component can only move horizontally or vertically. Now we want the component to also move diagonally.
- Create a keys array for the myGameArea object, and insert one element for each key that is pressed, and give it the value true , the value remains true untill the key is no longer pressed, the value becomes false in the keyup event listener function:
- **Example:**

Using The Mouse Cursor as a Controller

MoveByButton6.html

- If you want to control the red square by using the mouse cursor, add a method in myGameArea object that updates the x and y coordinates of the mouse cursor:
- **Example:**

```
var myGameArea = {
  canvas : document.createElement("canvas"),
  start : function() {
    this.canvas.width = 480;
    this.canvas.height = 270;
    this.canvas.style.cursor = "none"; //hide the original cursor
    this.context = this.canvas.getContext("2d");
    document.body.insertBefore(this.canvas, document.body.childNodes[0]);
    this.interval = setInterval(updateGameArea, 20);
    window.addEventListener('mousemove', function (e) {
      myGameArea.x = e.pageX;
      myGameArea.y = e.pageY;
    })
  },
  clear : function(){
    this.context.clearRect(0, 0, this.canvas.width, this.canvas.height);
  }
}
```

Example:

```
function updateGameArea() {
  myGameArea.clear();
  if (myGameArea.x && myGameArea.y) {
    blueFighter.x = myGameArea.x;
    blueFighter.y = myGameArea.y;
  }
  blueFighter.update();
}
```

Tracking Mouse Location

```
function updateGameArea() {
  myGameArea.clear();
  if (myGameArea.x && myGameArea.y) {
    blueFighter.x = myGameArea.x;
    blueFighter.y = myGameArea.y;
    var panel = document.getElementById("message_panel");
    panel.innerHTML = "["+blueFighter.x + ", "+ blueFighter.y + "]";
  }
  blueFighter.update();
}
</script>
<p id="message_panel"></p>
```

Touch The Screen to Control The Game

MoveByButton7.html

- We can also control the red square on a touch screen.
- Add a method in the myGameArea object that uses the x and y coordinates of where the screen is touched:
- **Example**

```
window.addEventListener('mousemove', function (e) {
myGameArea.x = e.pageX-25;
myGameArea.y = e.pageY-25;
myGameArea.z = true;
});
window.addEventListener('touchmove', function (e) {
myGameArea.x = e.touches[0].screenX-30;
myGameArea.y = e.touches[0].screenY-180;
myGameArea.z = true;
})
```

```
var myGameArea = {
  canvas : document.createElement("canvas"),
  start : function() {
    this.canvas.width = 480;
    this.canvas.height = 270;
    this.context = this.canvas.getContext("2d");
    document.body.insertBefore(this.canvas, document.body.childNodes[0]);
    this.interval = setInterval(updateGameArea, 20);
    window.addEventListener('touchmove', function (e) {
      myGameArea.x = e.touches[0].screenX;
      myGameArea.y = e.touches[0].screenY;
    })
  },
  clear : function(){
    this.context.clearRect(0, 0, this.canvas.width, this.canvas.height);
  }
}
```

Example

```
function updateGameArea() {
  myGameArea.clear();
  if (myGameArea.x && myGameArea.y) {
    blueFighter.x = myGameArea.x;
    blueFighter.y = myGameArea.y;
  }
  blueFighter.update();
}
```

```
if (myGameArea.z) {  // tracking by mouse or touch
  blueFighter.x = myGameArea.x;
  blueFighter.y = myGameArea.y;
  myGameArea.z = false;
}
```

Canvas Buttons

- We can also draw our own buttons on the canvas, and use them as controllers:
- **Example**

```
function startGame() {
  blueFighter = new component(30, 30, "red", 10, 120);
  myUpBtn = new button(30, 30, "blue", 50, 10);
  myDownBtn = new button(30, 30, "blue", 50, 70);
  myLeftBtn = new button(30, 30, "blue", 20, 40);
  myRightBtn = new button(30, 30, "blue", 80, 40);
  myGameArea.start();
}
```

Click Event

- Add a new function that figures out if a component, in this case a button, is clicked.
- Start by adding event listeners to check if a mouse button is clicked (mousedown and mouseup). To deal with touch screens, also add event listeners to check if the screen is clicked on (touchstart and touchend):

```
var myGameArea = {
  canvas : document.createElement("canvas"),
  start : function() {
    this.canvas.width = 480;
    this.canvas.height = 270;
    this.context = this.canvas.getContext("2d");
    document.body.insertBefore(this.canvas, document.body.childNodes[0]);
    this.interval = setInterval(updateGameArea, 20);
    window.addEventListener('mousedown', function (e) {
      myGameArea.x = e.pageX;
      myGameArea.y = e.pageY;
    })
    window.addEventListener('mouseup', function (e) {
      myGameArea.x = false;
      myGameArea.y = false;
    })
    window.addEventListener('touchstart', function (e) {
      myGameArea.x = e.pageX;
      myGameArea.y = e.pageY;
    })
    window.addEventListener('touchend', function (e) {
      myGameArea.x = false;
      myGameArea.y = false;
    })
  },
  clear : function(){
    this.context.clearRect(0, 0, this.canvas.width, this.canvas.height);
  }
}
```

Button Design

- Now the myGameArea object has properties that tells us the x- and y-coordinates of a click. We use these properties to check if the click was performed on one of our blue buttons.
- The new method is called clicked, it is a method of the component constructor, and it checks if the component is being clicked.
- In the updateGameArea function, we take the neccessarry actions if one of the blue buttons is clicked:

```
function button(width, height, color, x, y) {
  this.width = width;
  this.height = height;
  this.speedX = 0;
  this.speedY = 0;
  this.x = x;
  this.y = y;
  this.update = function() {
    ctx = myGameArea.context;
    ctx.fillStyle = color;
    ctx.fillRect(this.x, this.y, this.width, this.height);
  }
  this.clicked = function() {
    var myleft = this.x;
    var myright = this.x + (this.width);
    var mytop = this.y;
    var mybottom = this.y + (this.height);
    var clicked = true;
    if ((mybottom < myGameArea.y) || (mytop > myGameArea.y) || (myright < myGameArea.x) || (myleft >
myGameArea.x)) {
      clicked = false;
    }
    return clicked;
  }
}
```

Button Event Handlers

- Now the myGameArea object has properties that tells us the x- and y-coordinates of a click. We use these properties to check if the click was performed on one of our blue buttons.
- The new method is called clicked, it is a method of the component constructor, and it checks if the component is being clicked.
- In the updateGameArea function, we take the neccessarry actions if one of the blue buttons is clicked:

```
function updateGameArea() {
  myGameArea.clear();
  if (myGameArea.x && myGameArea.y) {
    if (myUpBtn.clicked()) {
      myGamePiece.y -= 1;
    }
    if (myDownBtn.clicked()) {
      myGamePiece.y += 1;
    }
    if (myLeftBtn.clicked()) {
      myGamePiece.x += -1;
    }
    if (myRightBtn.clicked()) {
      myGamePiece.x += 1;
    }
  }
  myUpBtn.update();
  myDownBtn.update();
  myLeftBtn.update();
  myRightBtn.update();
  myGamePiece.update();
}
```

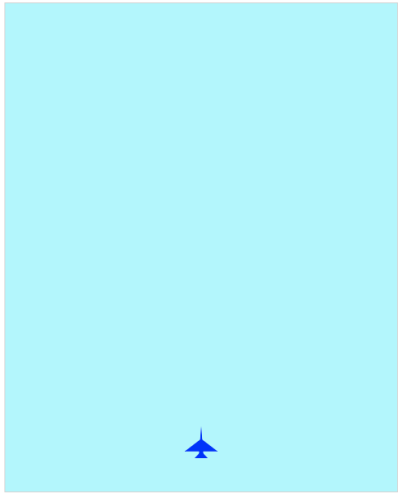
Game Scene Design

LECTURE 5

Scene Design

scene1.html

- scene1.html is a modified version of moveButton5.html
- Add background color and moving cloud.



- **Example: Background color**

```
function drawBackground(){
    var ctx = myGameArea.context;
    ctx.fillStyle = "rgb(174, 245, 252)";
    ctx.fillRect(0, 0, myGameArea.canvas.width, myGameArea.canvas.height);
}
```

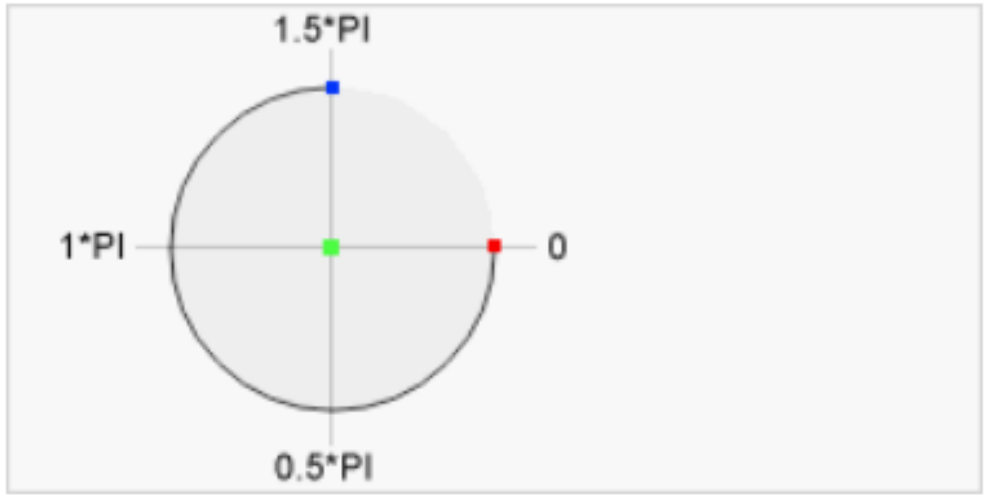
Draw Oval

```
function ellipse(ctx, cx, cy, rx, ry){
        ctx.save(); // save state
        ctx.beginPath();
        ctx.translate(cx-rx, cy-ry); // new origin
        ctx.scale(rx, ry); // new x, y scale
        ctx.arc(1, 1, 1, 0, 2 * Math.PI, false);
        ctx.restore(); // restore to original state
        ctx.fill();
}
```

Parameter Values

Parameter	Description
x	The x-coordinate of the center of the circle
y	The y-coordinate of the center of the circle
r	The radius of the circle
sAngle	The starting angle, in radians (0 is at the 3 o'clock position of the arc's circle)
eAngle	The ending angle, in radians
counterclockwise	Optional. Specifies whether the drawing should be counterclockwise or clockwise. False is default, and indicates clockwise, while true indicates counter-clockwise.

Definition and Usage



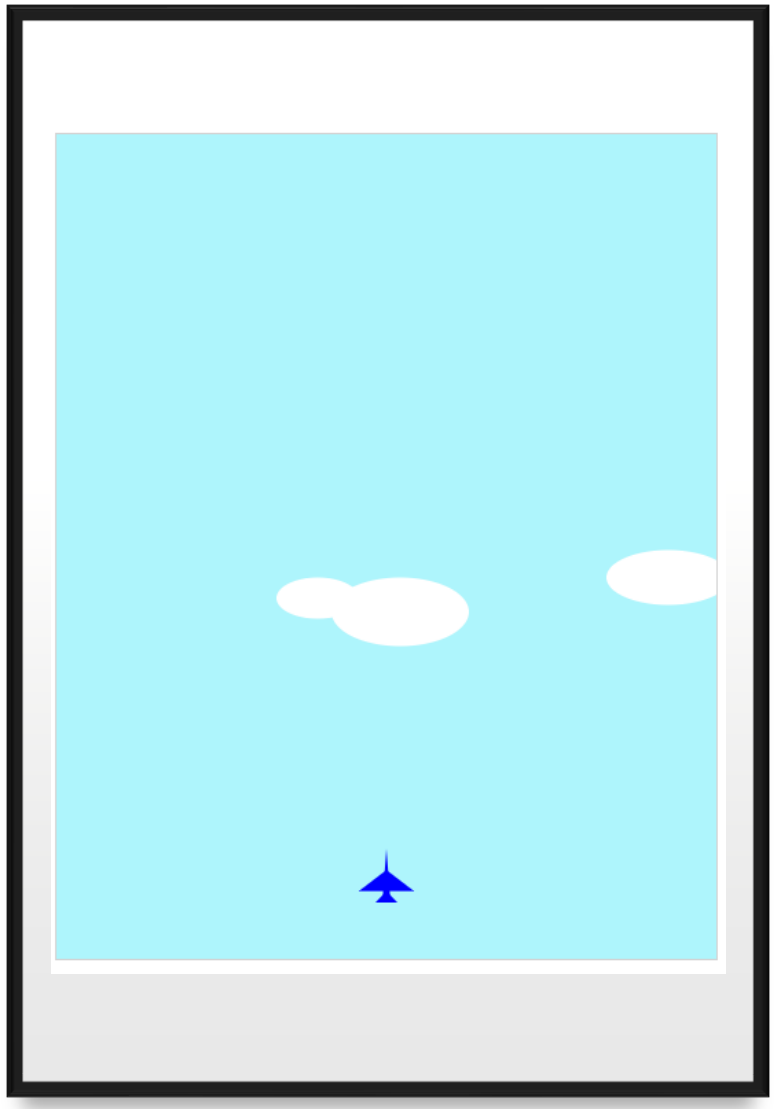
- Center arc(**100,75**,50,0*Math.PI,1.5*Math.PI)
- Start angle arc(100,75,50,**0**,1.5*Math.PI)
- End angle arc(100,75,50,0*Math.PI,**1.5*Math.PI**)

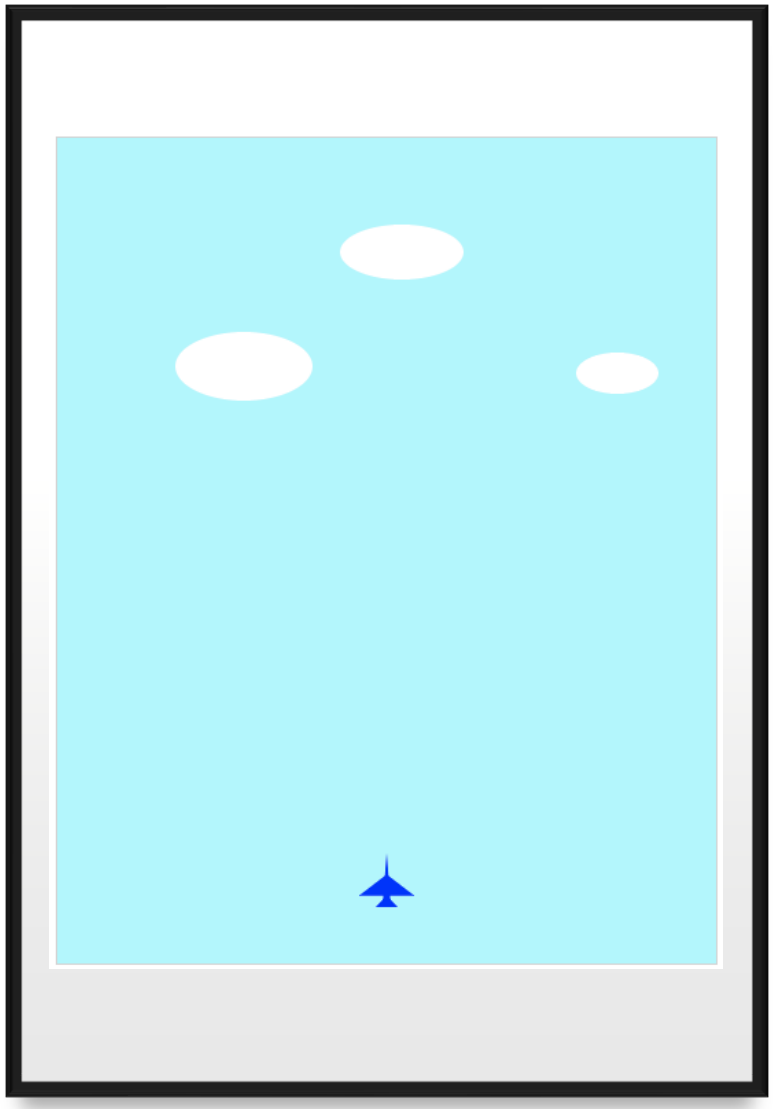
- The arc() method creates an arc/curve (used to create circles, or parts of circles).
- Tip: To create a circle with arc(): Set start angle to 0 and end angle to 2*Math.PI.
- Tip: Use the stroke() or the fill() method to actually draw the arc on the canvas.

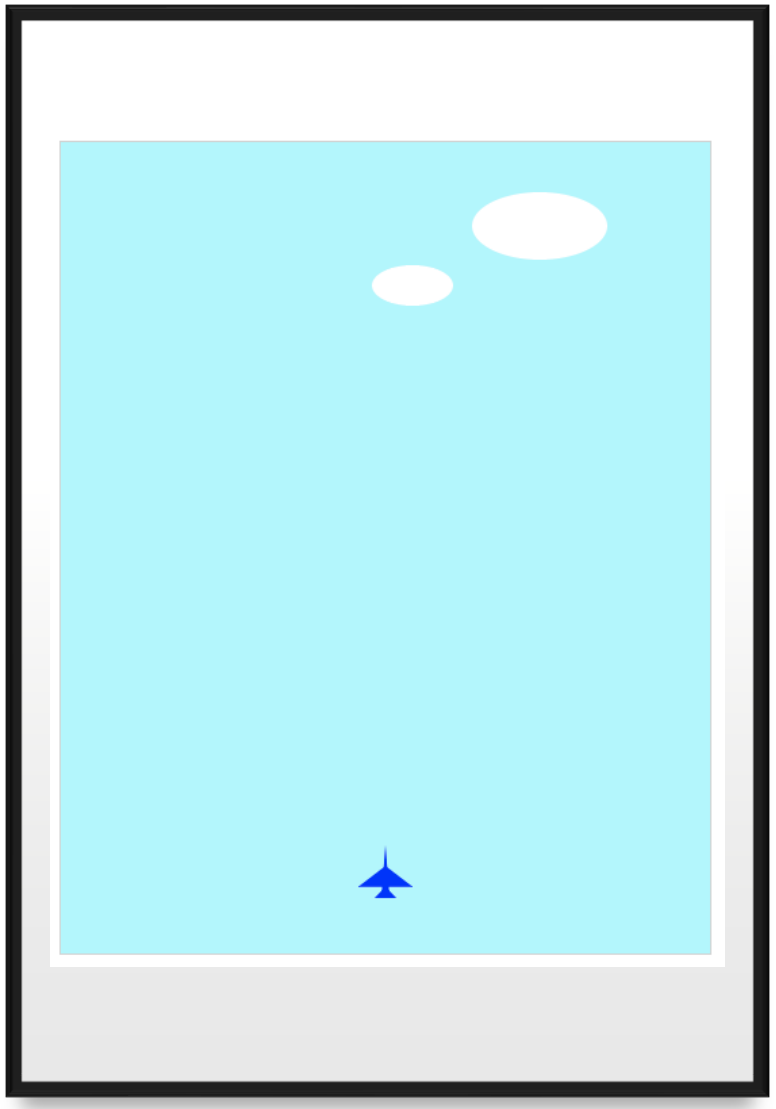
Oval Object

- The ellipse function is to draw an ellipse.
- This Oval constructor is a build a game component which will draw ellipse with newPos(), update() functions.
- The update() is to redraw the ellipse.
- The newPos() is the re-calculation of the new position for each tick. The oval object will re-enter the game area if it get out of bound.

```
function oval(width, height, color, x, y) {
    this.width = width;
    this.height = height;
    this.speedX = 0;
    this.speedY = 0.5;
    this.x = x;
    this.y = y;
    this.update = function() {
        ctx = myGameArea.context;
        ctx.fillStyle = color;
        ellipse(ctx, this.x+0.5*width, this.y+0.5*height, 0.5*width, 0.5*height);
    }
    this.newPos = function() {
        this.x += this.speedX;
        this.y += this.speedY;
        if (this.y>myGameArea.canvas.height+100) {
            this.x = Math.floor(Math.random()*500+20);
            this.y = Math.floor(Math.random()*50);
            this.width = Math.floor(Math.random()*20)+width-10;
            this.height = Math.floor(Math.random()*10)+height-5;
        }
    }
}
```



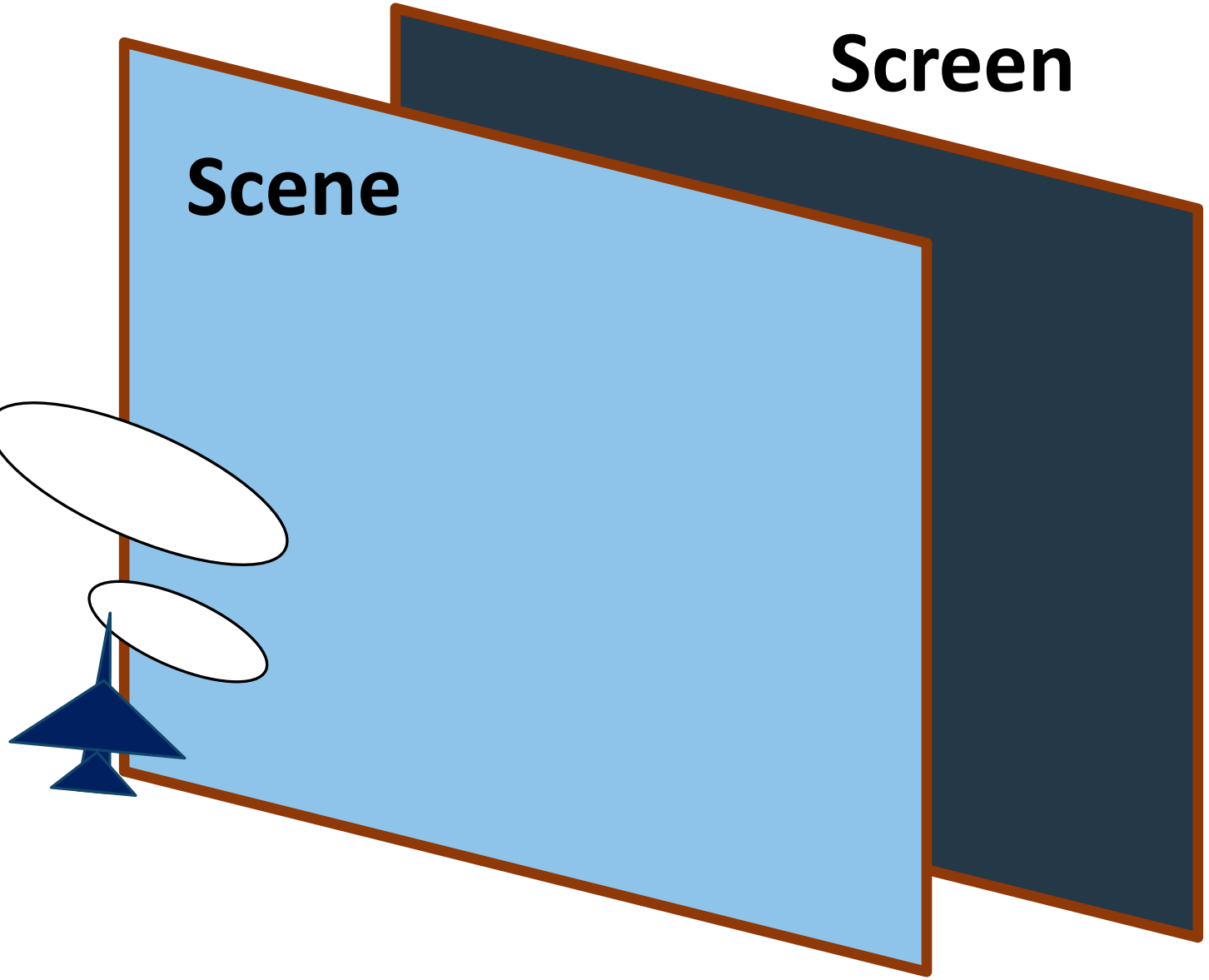




Components in the Scene

```
function startGame() {
    blueFighter = new plane(40, 40, "blue", 220, 520);
    ov1 = new oval(100, 50, "white", 200, 40);
    ov2 = new oval(60, 30, "white", 160, 40);
    ov3 = new oval(90, 40, "white", 400, 20);
    myGameArea.start();
}
```

```
function drawBackground(){
    var ctx = myGameArea.context;
    ctx.fillStyle = "rgb(174, 245, 252)";
    ctx.fillRect(0, 0, myGameArea.canvas.width, myGameArea.canvas.height);
    ov1.newPos();
    ov1.update();
    ov2.newPos();
    ov2.update();
    ov3.newPos();
    ov3.update();
}
 function updateGameArea() {
  myGameArea.clear();
  drawBackground();
  blueFighter.speedX = 0;
  blueFighter.speedY = 0;
  if (myGameArea.keys && myGameArea.keys[37]) {blueFighter.speedX = -1; }
  if (myGameArea.keys && myGameArea.keys[39]) {blueFighter.speedX = 1; }
  if (myGameArea.keys && myGameArea.keys[38]) {blueFighter.speedY = -1; }
  if (myGameArea.keys && myGameArea.keys[40]) {blueFighter.speedY = 1; }
  blueFighter.newPos();
  blueFighter.update();
}
```



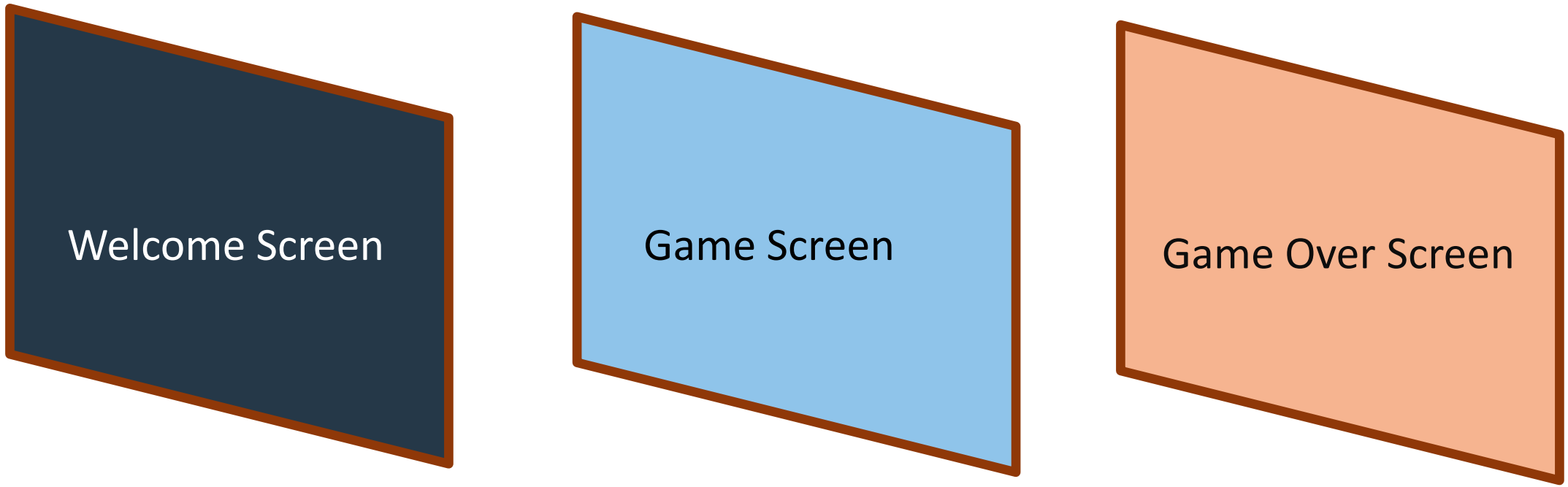
Components

- Plane
- Oval

Multiple Screen Design

scene2.html

- scene2.html is a modified version of scene1.html



Game Score and Winning Conditions

LECTURE 6

Art Design for a Game

LECTURE 6

Game Animation Design

LECTURE 6

CANVAS