The <canvas> Element and Writing Video Games

Contents

<canvas>

A brief introduction to <canvas> and video game functionality.

The <canvas> element creates a rectangular space on a web page for the devloper to manipulate JavaScript to render graphics, such as graphs and animations, as well as handle photo compositions and video processing.

<canvas> can actually be used to create video games to be played in a web browser. JavaScript code is used to build the game elements, such as a player character, an image background and game rules such as win conditions and collision detection, which display to the player through the canvas.

Preparing the <canvas> and JavaScript

This document instructs on how to prepare the proper environment to write a video game with <canvas> and JavaScript.

To accomplish this goal you will need to:

- Have an HTML document.
- Have a JavaScript file.
- *Know how to write in HTML.*
- *Know how to write in JavaScript.*

1. In the HTML document, create a <canvas> element, assigning it an ID, width and height.
2. In the HTML document, create a <script> element, providing the name of the JavaScript file as the source.
3. In the JavaScript file, write a variable named canvas that gets the ID assigned to the <canvas> in the HTML document.
4. In the JavaScript file, write a variable named ctx that gets a 2d drawing context for the variable canvas.

Displaying an image for a video game in <canvas>

This document instructs on how to load and render an image in a video game.

To accomplish this goal you will need to:

- *Have the <canvas> element and JavaScript file declared and called properly.*
- Provide an image for the JavaScript to load.
- *Write a function to render the image in the update loop.*

1. Load the image in JavaScript.
   a) Write variables for a new image and a true / false state for when the image is ready.
   b) Write an onload function that sets the true / false state to true.
   c) Set a source for the Image() method.
2. Create and call the rendering function.
   a) Declare the function.
   b) Refer to the <canvas> drawing context in a drawImage method.
3. Call the rendering function within the *update loop.*

Creating a character for a video game in <canvas>

This document instructs on how write a player character or an enemy character in a video game.

To accomplish this goal you will need to:

- *Have the <canvas> element and JavaScript file declared and called properly.*
- *Load an image to represent the character.*
- *Set up an object containing variables of the character and modify them.*

1. Create an object for the character.
   a) Write a name for the object to reference later, with an equals sign and a bracket.
   b) Write variables for coordinates (x, y), dimensions(width, height), vertical and horizontal velocity(xVelocity and yVelocity) and maximum speed (speed) along with a true / false state that determines if the player is jumping (isJumping), closing the bracket when finished.
   c) Create variables for friction and gravity.

2. Choose between designing a player character and an enemy character.

Character Type	**How to write**
Player character	*Follow this task to enable user input.*
Enemy character	*Follow this task to create enemy movement routine.*

Creating a player character for a video game in <canvas>

This document instructs on how write a player character or an enemy character in a video game.

To accomplish this goal you will need to:

- Set up an object containing variables of the player character and modify them.
- Choose to make the player character after setting up a character.

1. Enable user input by writing keys = [].

2. Modify velocity variables in response to player input inside the *update loop.*
   a) Write if statements for the keys that correspond to up, left and right: if (keys[X]){
   b) For the "up" if statement, set player.isJumping to true, set player.speed as negative and assign that value to player.yVelocity.
   c) For the "left" if statement, write an if statement that adds 1 to player.xVelocity if it is less than player.speed.
   d) For the "right" if statement, write an if statement that subtracts 1 from player.xVelocity if it is greater than player.speed.

3. Update the position variables based on the velocity variables in the update loop.
   a) Set player.xVelocity equal to its current value multiplied by the friction variable.
   b) Set player.yVelocity equal to its current value plus the gravity variable.
   c) Set player.x and player.y equal to player.xVelocity and player.yVelocity.
   d) Write a series of if statements checking if player.x is less than 0 or greater than the width of the canvas.
   e) Set player.x equal to the width of the canvas if it exceeds that value, or set player.x to 0 if it is less than 0..
   f) Write an if statement checking if player.y is less than 0.
   g) Set player.isJumping to false and player.y to 0 if it is less than 0.

Creating an enemy character for a video game in <canvas>

This document instructs on how write a player character or an enemy character in a video game.

To accomplish this goal you will need to:

- Set up an object containing variables of the player character and modify them.
- Choose to make an enemy character after setting up a character.

1. Write if statements in the *update loop.* to tell the enemy to turn around when it reaches an edge of the canvas.
   a) Write a series of if statements checking if enemy.x is less than 0 or greater than the width of the canvas.
   b) Set enemy.x equal to the width of the canvas if it exceeds that value, and set an enemy.faceLeft boolean value to true.
   c) Set enemy.x equal to 0 if it is less than 0, and set enemy.faceLeft to false.
2. Create a movement pattern for the enemy to walk back and forth in the update loop.
   a) Set enemy.x equal to its current value plus its speed value to move to the right.
   b) Set enemy.x equal to its current value minus its speed value to move to the left.

JavaScript Conventions

A table of some basic JavaScript writing conventions for writing variables, functions and values.

JavaScript element	How to write
Variables	var isReady = false;
Functions	functionName = function(){}
Actions on variables	variableName.actionName = function(){}
Values	width = 640;
Character object	objectName = {}
Character values	width: 28, jumping: false;
Reference character values outside of character object	objectName.valueName;
Enable user input	keys = []
Keyboard values	keys[38] = up arrow, keys[39] = right arrow, keys[37] = left arrow, keys[32] = space
If statements	if (isReady){}, if (!isReady){}
Operations on numeric values	valueName += 7, valueName -= 2

HTML Conventions

A table of some basic HTML writing conventions for writing elements and values.

HTML code	How to write
Element ID	<canvas id="a"></canvas>
Inline value / style	<canvas width="640" height="480"></canvas>
External source file	<script src="game.js"></canvas>

Update Loops in JavaScript

The update loop enables a JavaScript application to change states over time, which is essential for most video games.

An update loop, used in games, is JavaScript code that handles computations of game variables and draws those results to the canvas using pre-defined rendering functions. This enables functionality like movement of player and enemy characters, adjustments to the states of those characters such as health and power-ups, animation of background and character images and win conditions. Typically, an update loop is made up of a few parts:

- A function defining a variable called requestAnimationFrame set equal to one of a few window.requestAnimationFrame variables based on what browser is in use, with window.requestAnimationFrame being set to requestAnimationFrame
- A function named update that serves as the main body of code for the game, with a call of requestAnimationFrame on itself at the end
- A trio of addEventListener functions that respond to user input on the keyboard.

requestAnimationFrame

```
(function() { var requestAnimationFrame = window.requestAnimationFrame || window.mozRequestAnimationFrame
|| window.webkitRequestAnimationFrame || window.msRequestAnimationFrame; window.requestAnimationFrame =
requestAnimationFrame;})();
```

update()

```
function update(){ requestAnimationFrame(update);}
```

addEventListener

```
document.body.addEventListener("keydown", function(e) { keys[e.keyCode] = true;});
document.body.addEventListener("keyup", function(e) { keys[e.keyCode] = false;});
window.addEventListener("load",function(){ update();});
```