**Basic Functionality**

In this game you have a Player and Enemies (bugs). The goal of the player is to reach the water, without colliding into any one of the enemies.

* ~~The player can move left, right, up and down~~
* ~~The enemies move at varying speeds on the paved block portion of the game board~~
* ~~Once a the player collides with an enemy, the game is reset and the player moves back to the starting square~~
* ~~Once the player reaches the water (i.e., the top of the game board), the game is won~~

Check out a quick video demo of the game in action below!

**Additional Functionality (Optional)**

In addition to the basic functionality, you can add more cool functionality to your game. For example, here are some additional features that you can add:

* Player selection: allow the user to select the image for the player character before starting the game. You can use the different character images provided in the images folder (we’ll get to that below)
* ~~Score: you can implement a score for the game. For example, the score can increase each time the player reaches the water, and it can be reset to 0 when collision occurs (or it can be reduced)~~
* Collectibles: you can add gems to the game, allowing the player to collect them to make the game more interesting
* Anything else you like!

**Development Strategy**

Inside the app.js file, you will need to implement the Player and the Enemy classes, using [Object-Oriented JavaScript](https://www.udacity.com/course/object-oriented-javascript--ud711). Be sure to review *all* code comments written in app.js. Part of the code for the Enemy is provided to you, and you will need to complete the following:

* The Enemy function, which initiates the Enemy by:
  + ~~Loading the image by setting this.sprite to the appropriate image in the image folder (already provided)~~
  + ~~Setting the Enemy initial location (you need to implement)~~
  + ~~Setting the Enemy speed (you need to implement)~~
* The update method for the Enemy:
  + ~~Updates the Enemy location (you need to implement)~~
  + ~~Handles collision with the Player (you need to implement)~~

You can add your own Enemy methods as needed.

You will also need to implement the Player class, and you can use the Enemy class as an example on how to get started. At minimum you should implement the following:

* The Player function, which initiates the Player by:
  + ~~Loading the image by setting this.sprite to the appropriate image in the image folder (use the code from the Enemy function as an example on how to do that)~~
  + ~~Setting the Player initial location~~
* The update method for the Player (can be similar to the one for the Enemy)
* ~~The render method for the Player (use the code from the render method for the Enemy)~~
* ~~The handleInput method, which should receive user input, allowedKeys (the key which was pressed) and move the player according to that input. In particular:~~
  + ~~Left key should move the player to the left, right key to the right, up should move the player up and down should move the player down~~
  + ~~Recall that the player cannot move off screen (so you will need to check for that and handle appropriately)~~
  + ~~If the player reaches the water the game should be reset by moving the player back to the initial location (you can write a separate reset Player method to handle that)~~

You can add your own Player methods as needed as well. Once you have completed implementing the Player and Enemy, you should instantiate them by:

* ~~Creating a new Player object~~
* ~~Creating several new Enemies objects and placing them in an array called allEnemies~~

Beyond that, feel free to add additional functionality to the game. You can add more code to the app.js file and to the Enemy and Player classes to accomplish that.

# HTML5 Canvas Info

The starting code for the Classic Arcade Game Clone project handles most of the drawing for you. The <canvas> element has already been created and the two-dimensional drawing context for the canvas element is available as the ctx object in the app.js file.

**Drawing an Image**

In the app.js file, you can see in the Enemy class. This class has a render() method that uses the ctx.drawImage() method. This method takes three parameters: an image, an x-coordinate, and a y-coordinate:

ctx.drawImage(Resources.get(**this**.sprite), **this**.x, **this**.y);

**Available Images**

In this example, the game engine has a Resources object that caches all of the images needed for the game so you don’t have to wait for them to load during gameplay. The images available to use are listed in engine.js:

Resources.load([

'images/stone-block.png',

'images/water-block.png',

'images/grass-block.png',

'images/enemy-bug.png',

'images/char-boy.png'

]);

There are many other images available with the starter code. If you want to use them in your game, all you need to do is include them in the array passed to the Resources.load() method in engine.js near the bottom of the file:

Resources.load([

'images/stone-block.png',

'images/water-block.png',

'images/grass-block.png',

'images/enemy-bug.png',

'images/char-boy.png',

'images/char-pink-girl.png'

]);

**Expanding on the Existing Capabilities?**

It’s unlikely that you’ll need any additional methods of the ctx object for the project. However, if you decide to add additional features to your game, you might want to incorporate some of these methods. If you do, the ctx object is a CanvasRenderingContext2D object. [This documentation](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D) provides all of the methods that are available to that object.