**Project Rubric**

Your project will be evaluated by a Udacity code reviewer according to the [Classic Arcade Game Clone project rubric](https://review.udacity.com/#!/rubrics/15/view). Please review for detailed project requirements.

**Overview: Game Functionality**

Your implementation must at minimum follow the basic functionality, but you can add additional optional functionality to your game, if you wish.

**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!

**First, Get the Starter Code**

You won’t have to build the game from scratch. We have provided the [art assets and game engine](https://github.com/udacity/frontend-nanodegree-arcade-game) for you. You can download or clone them from our repository.

The repository contains css, images, and js folders, as well as an index.htmland a README.md file.

* The css folder contains a style.css file which you do not need to edit
* The images folder contains the png image files, which are used when displaying the game. The images for the player and enemy character are going to be loaded from this folder.
* The js folder also contains the app engine needed to run the game and a resources.js file. You do not need to edit these files.
* Opening index.html should load the game
* The README.md file should contain instructions on how to load and play the game (you will need to add those instructions).

Once you have downloaded the files we have provided, you will have to edit app.jsto build the game.

**Object-Oriented JavaScript**

Great! You have the starter code. But before moving on, make sure you are comfortable with the content from [Object-Oriented JavaScript](https://www.udacity.com/course/object-oriented-javascript--ud711). Ask yourself:

* What is an **object**?
  + How is an object different from a **primitive** in JavaScript (e.g., can a primitive have methods?)
  + What are some ways to create or instantiate a new object?
  + How do you modify **properties**, or add/remove properties from an object?
* What is a **constructor function** (or class)?
* What is this? What does it refer to and how is it used in different contexts (i.e., in a constructor function, a **method**, etc.)?
* How do you add a property or method to a constructor's prototype?
* What is **prototypal inheritance** and how is it implemented?

**Development Strategy**

Inside the app.js file, you will need to implement the Player and the Enemyclasses, 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.

**Still Not Sure How to Begin?**

Let's recap how you can approach this project:

1. If you need a refresher on [Object-Oriented JavaScript](https://classroom.udacity.com/courses/ud711), review our course and [OOJS Notes](https://docs.google.com/document/d/1F9DY2TtWbI29KSEIot1WXRqqao7OCd7OOC2W3oubSmc/pub?embedded=true)
2. Above are a few notes on HTML5 Canvas, but if you'd like a more detailed explanation as to how the game engine works, see our [HTML5 Canvas](https://www.udacity.com/course/ud292) course
3. Review how the game functions. How is it played? What does the user get to control? What are the winning and losing conditions?
4. Download the [art assets and provided game engine](https://github.com/udacity/frontend-nanodegree-arcade-game) (starter code)
5. Review *all* code and comments provided in app.js
6. Identify the various classes you will need to write
7. Identify and code the properties that each class must have to accomplish its tasks
8. Write the functions that provide functionality to each of your class instances
9. Review the [project rubric](https://review.udacity.com/#!/rubrics/15/view) to make sure your project is up to spec. For example make sure the functions you write are *object-oriented*; either class functions (like Player and Enemy) from ES6 or class prototype functions such as Enemy.prototype.checkCollisions, and that the keyword this is used appropriately within your class and class prototype functions. Also be sure that the README.md file is updated with your instructions on both how to run *and* play your arcade game
10. You are welcome to use ES6 to build your classes

For further research, check out these [additional resources](https://knowledge.udacity.com/questions/4580) in Knowledge, curated by students. Feel free to review some external resources below:

* [HTML5 Canvas Game: 2D Collision Detection](http://blog.sklambert.com/html5-canvas-game-2d-collision-detection#d-collision-detection)
* [Adding collision detection to images drawn on canvas](https://stackoverflow.com/questions/13916966/adding-collision-detection-to-images-drawn-on-canvas) on StackOverflow
* [Modals in Pure ES6 JavaScript](https://lowrey.me/modals-in-pure-es6-javascript/)
* [KeyboardEvent Value (keyCodes, metaKey, etc)](https://css-tricks.com/snippets/javascript/javascript-keycodes/) via CSS-Tricks