Incremental Game Engine JS – v1.0 Tutorial

Goal: http://aldo111.github.io/incremental-game-engine-js/tutorial/demo.html

[1.0] Preface

So you want to make an incremental game using Javascript? Great! I would just like to mention that there are a few other engines out there. Some rely on only data and others give you a visual editor to work with. In both cases, there's no coding involved. This **open-source** engine is here to provide not just an alternative to those (great) options, but to also encourage people to take up coding and expand upon the core engine and create wonderful things!

Feel free to jump to 2.0 directly after you get the zip folder!

[1.1] Getting Started

(If you want to skip this stuff about 'git' or aren't interested in learning a bit about version control, you can get started by downloading this: https://github.com/Aldo111/incremental-game-engine-js/archive/tutorial.zip and heading to 1.2)

To maintain this project, we utilize a ***version control system*** **(VCS)** called **git**. A VCS allows people to collaborate over projects in efficient and useful ways.

In **git**, the analogy is to that of a tree: just how every tree has branches, so too in git every project is a tree and it has its own branches. One of these branches, the very first branch to come out of this tree, is called **the master branch**. This branch stores the primary version of the project directory (otherwise known as a **repository**).

This tree that we use is really dynamic. For example, if multiple people are working on the same project and each of them want to add their own feature – they can do it by simply creating another 'branch' based on the master branch, in the 'tree' of the project. They can then continue making changes to this new branch while leaving the master branch undisturbed until they're ready to merge their branch with it!

So in order to make this tutorial really easy for you, we've created a 'tutorial branch'.

How can you get access to this tutorial branch on your computer?

git clone -b tutorial https://github.com/Aldo111/incremental-game-engine-js.git

Once you run the above in your command line interface, you should be good to go.

Alternatively, you can download this:
https://github.com/Aldo111/incremental-game-engine-js/archive/tutorial.zip

[1.2] Planning Our Game!

First, we need to plan our game. This stage of the game development workflow is simple: we come up with a basic idea/premise for our game, a goal to work towards and then we can start!

For the purposes of this tutorial (and to get essential concepts of the IGE across), let's keep things VERY simple:

Premise: The player has 500 Credits and has decided to become a millionaire by buying houses and earning money from it!
Goal: To earn a million credits.

Great, now we have something to start with and flesh out. But before we actually begin coding (I know, this is all boring but I promise we're almost there!), it's important to understand what our premise and our goal means **technically-speaking**.

So let's...

[1.3] Break Things Down

Before we begin even a simple implementation of our game it's important to understand what our premise and goal translate to behind-the-scenes.

So let's re-examine our Premise and Goal and see what we can make out from it.

From the Premise:
I) There is an attribute, otherwise called a **variable**, which tracks the player's money
2) The player is buying and selling cars, so we need to create and store a list of cars!
3) Once the variable that stores the player's money has reached 1000000 (a million), he's basically hit our endgame.

Alright, that was quick and painless, wasn't it? Now let's actually make this game!

[2.0] One Directory

Alright, let's look at what we have in this tutorial folder!
We have 2 folders and 1 file.
a) **css/ –** stores our stylesheets/design for the web page
b) **js/ -** stores our game engine script along with a jquery script.

c) **index.html** – a simple html template to get started with.

We will only work in index.html. So get out your favorite text editor (Notepad, Notepad++, TextWrangler, Sublime..etc)

[2.1] Setting up our code flow

In index.html, you should see something like the following:

```
1 <script>
2
3                 $(document).ready(function() {
4
5                 });
6
7 </script>
```

Line 3 is jQuery (a javascript library that makes a lot of things much easier) that simply executes the code in between its curly brackets once the page is ready/loaded!

This is where we shall place some **initialization** code.

Let's decide what kind of functions and variables we need:
a) A variable to store the Game state (var game)
b) A function to initialize the Game and its variables (function init())
c) A function that contains our main incremental-game-related code (function incrementalGame())
d) A function that executes the above in a loop (function play())

*The above are all **essential** functions in order to create our game and get it running. Now let's think of slightly non-essential functions.*

e) A function that does something when we decide to buy a house (function buyHouse(...))

```
1  <script>
2  //declare variables and other data here
3                  var game;
4
5                  $(document).ready(function() {
6                   init(); //call our initialization function
7                  });
8
9
10
11 </script>
```

Line 4 declares our main game variable.
Line 7 invokes a function called **init()** which, as we said earlier, will simply initialize our game and other data.

Now let's make some functions!

[2.2] Initializing data and the init() function

Apart from the incrementalGame() function, the init() function is really where the core of our game is established.

Here is what we need to do here:

1. Create data for our houses
2. Store the data somewhere
3. Initialize the game with our data
4. Add some html elements to make things into which we can print the houses and add a clicker button
5. We're done here!

So let's do it:

1. Create Data and 2. Store the Data somewhere

```
<script>
//declare variables and other data here
                var game;

                //our data for a set of houses
                var shackOptions={type:"Shack", perSecondModifier:100, cost:500}; //a shack object that
gives 500
                var mansionOptions={type:"Mansion", perSecondModifier: 200, cost:25000};
                var estateOptions={type:"Estate",perSecondModifier: 1000, cost:100000};
                var castleOptions={type:"Castle",perSecondModifier: 10000, cost:300000};


                $(document).ready(function() {
                 init(); //call our initialization function
                });



</script>
```

Here we're simply adding data in the form of Javascript Objects!

You'll see why soon enough, and you'll see if there's an alternate way to store this data.

3. Initialize the game with our data

```
1  function init() {
2
3          game=new Game(30);//30 fps game
4          game.init();//binds a clicker element -> useful for games where single-variable clicker games like co
5          game.addToScore(500);//initialize score to 500
6         game.addToPointsPerClick(10);//initialize points received per click to 10
7
8         game.addSet("HOUSES");//create a game set that stores all the Houses the player can buy
9
10        //add the houses into the HOUSES entity set
11        game.sets.HOUSES.addEntity(“Old Shack”, shackOptions);
12        game.sets.HOUSES.addEntity("Mansion",mansionOptions);
13        game.sets.HOUSES.addEntity("Estate", estateOptions);
14        game.sets.HOUSES.addEntity("Castle", castleOptions);
15 }
```

Let's go over this line by line

function init() → creates our function to initialize stuff

Lines 3-6 →
game=new Game(30); initializes a game that runs at 30FPS (Frames per second).
If you simply did **game =new Game();// then FPS would default to 30.**

game.init(); → is a Game function that basically detects clicks on any html element with the id “clicker” and accordingly changes Game's internal **score** variable by **x** number of points, where **x** is another internal variable which can be seen by **game.getPointsPerClick();**
This value (pointsPerClick) starts out at 0, like all other internal number variables (there are 3 : score, pointsPerClick, pointsPerSecond).

Game.addToScore(500); → initializes the internal score variable to 500.
Game.addToPointsPerClick(10); → initializes the internal (points received per click) variable to 10. So now every time the user clicks the #clicker element, the user gets +10 to his score.

We don't do anything with pointsPerSecond (although there is a function for that too called addToPointsPerSecond(value)) simply because we don't want the user gaining any money initially.

Line 8 →
game.addSet(“HOUSES”);

This creates a set or list in our Game called “HOUSES”. So it is in this list where we will store our house data.

To access any set in the Game() object, you simply need to do **game.sets.<name>**
Every set has a name and a list of items known as **entities**.
For example: **game.sets.HOUSES.getName()** → will give us "houses"
game.sets.HOUSES.getSet() → will give us its internal list of items

Suppose there is an entity called "House", which has an **attribute** called 'cost'
game.sets.HOUSES.getSet().House.attributes.cost; → gives us the value of cost from this entities internal list of attributes.
While it might seem a bit too long, it does help keep things organized and easy to read/understand. Plus you can always short things by doing:

```
var list=game.sets.HOUSES.getSet();
alert(list.House.attributes.cost);
```

One thing to keep in mind is that EVERY EntitySet and every Entity must have a unique name upon creation. Think of this as its own unique identifier that allows the Game() code to quickly find the entity or entity set you want and return it!

Lines 11 – 14 →

Now we're just adding Entities into the HOUSES set by **game.sets.HOUSES.addEntity(<identifier>, <attributes javascript object>);**

Alternatively, we could have also done:

```
var Shack=new Entity("Old Shack", {type: "Shack", perSecondModifier: 100, cost: 500} );
//creates an entity
game.sets.HOUSES.addEntity(shack);
```

Every entity has two main variables:
a name and a list of attributes. This list of attributes is in the same form as the options you supply in the above code – javascript object notation.

So if we have an entity called **Shack** like the above code, then we can get its attributes like this:
alert("Cost of a Shack:" **+ Shack.attributes.cost);**

Quick and painless!

NOTE: Game() comes with a function called Game.size(<associative array>) which returns the size of the list passed to it. These lists include the Game.sets, Game.attributes, EntitySet.getSet(), Entity.attributes

Make sure you familiarize yourself with this portion of the code as this was really the main work! Chances are in the future things will change and it may become even easier to do things, but for now this is how the Entities system works! Feel free to check out the JS file and see what else you can do with the current system.

4. Now all we do is add some HTML elements to display the list of houses, our clicker, our stats, and add interaction with some of these elements.
Go ahead and setup the HTML this way (I've already styled it in the css but you can change things around if you want!)

```
    <body>
1
2               <header>
3               GOLD HOUSE
4               </header>
5
6               <div id="available_houses">
7               </div>
8
9               <div id="money">
10              <b>Money</b>: <span id="money_value"></span><br>
11
12              <div id="clicker">
13              Do your regular job
14              </div>
15              <br>
16              Per Second: <span id="perSecond"></span><br>
17              Per Click: <span id="perClick"></span>
18              </div>
19
20
21
        </body>
```

There's no real explanation necessary for this section. All we're doing is just displaying stuff in html. Do note the clicker however; this is our main way of interacting with the score Variable.

Now back to the ***init() function:***

```
function init() {



        game.sets.HOUSES.addEntity("Castle", castleOptions);
//in addition to our previous code, add the stuff below..
        //now let's add this list into our html

        houses_list=game.sets.HOUSES.getSet();//obtain the set of houses
        for (j in houses_list)
        {
            $("#available_houses").append("<div class='house' onclick=\"buyHouse('"+j+"')\"><b>"+
houses_list[j].getName()+"</b><br>Cost: <b>"+houses_list[j].attributes.cost);



        }


                play();
            };
```

All we're really doing here is storing the getSet() value, or the list of houses, in a variable called **houses_list**.

Then we're iterating over this list by doing **for (j in houses_list)** where **j** is the identifier of that list entity (hence why we need to keep them unique!).

Inside the for-loop, all we're really doing using jQuery is adding our list of houses in a nice graphical form and also, for each div element that holds a house, we add an **onclick event** that invokes a function called **buyHouse(<house identifier>)** which handles what to do when we want to buy a house.

Finally, we invoke the **play()** function, which really just begins our game!

[2.3] buyHouse(<house identifier>)

```
1  function buyHouse(housey) {
2                       var house=game.sets.HOUSES.getSet()[housey].attributes;
3                       if (house.cost <= game.getScore())
4                       {
5
6                           game.addToScore(-house.cost);//subtract house cost from the score variable
7                           game.addToPointsPerSecond(house.perSecondModifier);
8                           game.addToPointsPerClick(house.perSecondModifier/100);
9
10                      }
11
12                  };
```

This is pretty straightforward. We passed the house identifier to this function and obtain that particular house entity's attributes. Then we check if the house can be bought based on the user's current game score, and then accordingly we modify:
a) the score
b) the points received per second (since the user has a new house, more money is earned)
c) the points received per click (since the user has a new house, perhaps he got promoted or just started his own company, who knows, he still gets more money from his job)

This function has two salient features, which I must point out even if you already observed them:
1) game.sets.HOUSES.getSet()**[housey]**
So far in my examples, I've used getSet().someName but over here we use square brackets. This is because some of our house identifiers may have spaces in their names so you can't really reference those hosues directly with the dot format (eg. HOUSES.Old Shack) so we use the square/array notation just to be safe in such a case. (Array notation works for any object)

2) I mentioned this earlier as well, but Game() internally has 3 defined variables called score, pointsPerSecond, and pointsPerClick, each with accessor functions (getScore, getPointsPerSecond...etc), and mutator functions (game.addToScore, game.addTo..etc)

This makes this engine ideal for single-variable games like cookie clicker or the demo we're whipping up. Things get slightly more complicated when we have multiple 'clickers' but this issue can be taken care of on the user's end by defining our own onClick events (like we did for the houses – the house buying mechanism turns into its own clicker mechanism when we start earning more money than we can spend).

[2.4] play()

```
function play() {

                setInterval(function() {

                    //game code
                    incrementalGame();



                },1000/game.getFPS());


        };
```

The play function essentially runs our game loop once we have finish initializing and setting up our game. We use a simple setInterval here which takes two parameters: a function and the periodic interval time (in **ms – 1000ms makes 1 second**) after which the function repeats itself.

Here we see the concept of FPS – Frames Per Second. So what we're doing here is **processing the game code 30 times per second.** This is not really necessary and most clicker games can get by with just 1 frame per second. Usually the use of FPS is for cosmetic purposes (if you have HTML5 canvas elements) or in our and Cookie Clicker's case, to show number's changing in between seconds as well – it just gives a nicer feeling to see your numbers go up incredibly quickly as opposed to waiting every second.

Internally, the Game() variable correctly handles the increase of score per second by dividing it by the FPS.
Finally, incrementalGame() :

```
function incrementalGame() {
                //our game code

                //display stuff
                $("#money_value").html(game.getScore());
                $("#perSecond").html(game.getPointsPerSecond());
                $("#perClick").html(game.getPointsPerClick());

                //do stuff
                game.increaseScorePerSecond();//this increases the score per second
                                              //using game's internal 'score' variable

        };
```

This really just displays and updates our various variables, and increases the score per second.

5. WE'RE DONE!

Your game should be up and running!

Feel free to tinker around with it, see what you can do more easily (or more difficultly) with the help of this engine. We've got a decent game up and running that really just needs data and stylistic code changes to turn into an even greater game, like adding new upgrades to the entity list after the user has reached a certain milestone and so on!

If you encounter any issues, make sure to check your javascript console and also check the online v1.0 version of this demo here for comparisons sake: **http://aldo111.github.io/incremental-game-engine-js/tutorial/demo.html**

Thank you for reading this and trying out Version 1.0 of the Incremental Game Engine JS. More (hopefully better) tutorials will come as more changes are made and more uses are discovered from this framework!