JsPhysics

Index

Setting up JsPhysics

Downloading the Library

By the GitHub Website

– Go to https://GitHub.com/MatheusTomazella/JsPhysics;

– Press the green button named “Clone or Download” and select “Download ZIP”;

– Go to where you downloaded (by default the folder “Download”);

– Right click on the ZIP folder and select “Extract Here”;

– Drag and drop the file “JsPhysics.js” to the root of your page directory.

By GitBash

– Go to your page directory and right click the blank space;

– Select “GitBash Here”;

– Type “Git clone https://github.com/matheustomazella/jsphysics.git” and press enter;

Important:

Make sure the file “JsPhysics.js” is on the root of your page’s directory.

Adding to the project

– Open your HTML file and add the library on the head tag by the code: “<script src=”JsPhysics.js”>”;

Initializing script

– On a new script tag, type “difineArea(canvasId);

– Make sure to add the quotation marks on the Id if it isn’t a variable;

– Add the function “Loop()” and start programming!

Introduction

PhysicalObjects System

All the library is based in a proper object type named PhysicalObject. It’s a simple object created by the user to keep all the information about a body.

Here’s its base structure:

```
name: {

    name: name,
    color: drawingColor,
    x: XaxisPosition,
    y: YaxisPosition,
    w: drawingWidth,
    h: drawingHeight,
    vx: velocityX,
    vy: velocityY,
    ax: accelerationX,
    ay: accelerationY,
    m: mass,
    bp: bouncingPercentage,
    gravity: T/F,
    collision: T/F

}
```

There is also another object that keeps the data related to the environment of the canvas:

```
globalConstants = {

    gravitationalacceleration: -0.2,
    backgroundcolor: "white",
    display: { h: 150, w: 300 }

}
```

All the object have their properties open to be changed by the user, so fell free to change, for example, the backgroundcolor propriety.

Velocity, acceleration and gravity

The canvas Y axis has higher values the lower the position, but the proprieties of velocity and acceleration use the more intuitive system, so a positive velocity makes the body go up.

Each PhysicalObject has proprieties related to velocity and acceleration. The acceleration proprieties increment their respective velocities each frame, and the velocities the position ones.

PhysicalObjects also have a property called gravity which keeps a boolean value. If it’s “true” the Y axis velocity is incremented by the value of gravity, saved on the globalConstants object.

Library functions

Physical Objects

Creating PhysicalObejcts

To create a PhysicalObject you have two options:

- **Creating manually**

Use pysicalObjects[obejctName] = { propriety1: value, property2: value... }

Always make sure to use the default parameters, other way it may cause some problems, but you still can add custom ones.

Example:

physicalObjects[“test”] = { name: ‘test’, color: ‘black’, x: 20, y: 75, w: 10, h: 10, gravity: true, collision: true };

- **Creating with the function createPhysicalObject( )**

Use the function createPhysicalObject( ) and pass as parameters the following sequence: objectName, color, initialPosX, initialPosY, drawingWidth, drawingHeight, useGravity, useCollision.

Example:

createPhysicalObject(‘test’, ‘black’, 20, 75, 10, 10, true, true);

Dealing with PhysicalObject’s proprieties

You can access all of the PhysicalObject’s properties by using:

physicalObjects.objectName.property = newValue

Or

physicalObjects[objectName].property = newValue

Example:

physicalObjects.test.vx = -20;

Or

physicalObejcts[“test”].vx = -20;

Deleting PhysicalObjects

To delete a PhysicalObject you can use:

delete physicalObjects.objectName

Or

deletePhysicalObject(objectName **)**

Example:

deletePhysicalObejct(‘test’);

Input

Adding inputs

To add an input you can use the function addInput( ). Inform the parameters keyCode(int), pressingFunction and releasingFunction(optional).

Example:

```
function goUp( ){ }
function stopGoUp( ){ }

addInput( 38, goUp, stopGoUp );
```

All of the inputs are saved on an object called inputs and can be accessed.

The eventListener for the keyboard inputs are already defined.

Collision

Adding collisions

To add a collision use the function createCollisionHandler( ) using the parameters PhysicalObject1, PhysicalObject2 and collisionFunction.

Example:

```
function stopAll( ){ }

createCollisionHandler( physicalObjects.test1, physicalObjects.test2, stopAll }
```

The object collisionHandlers keeps all of the collision situations created and is open to changes.

Loop

Adding functions to Loop

Loop is the function that initiate the simulation and keeps calling itself, so every function that needs to be repeated each frame has to be added to it.

The function addToLoop( ) use as parameter a function and a priority which can be either ‘high’ or ‘low’. A high priority function will run first than the others, and a low priority will run after all of them. The default to priority is ‘low’.

Example:

```
function score( ){ }

addToLoop( score, ‘high’ );
```