

## Acknowledgement

We take immense pleasure in the successful completion of **Asteroid Game Engine**. The environment at Padre Concêncio College of Engineering, Goa greatly facilitated the completion of this project.

We greatly appreciate the support, encouragement, and understanding extended towards our project by our guide Prof. **Andrea D'Souza** and co-guide Prof. **Cynara Fernandes**.

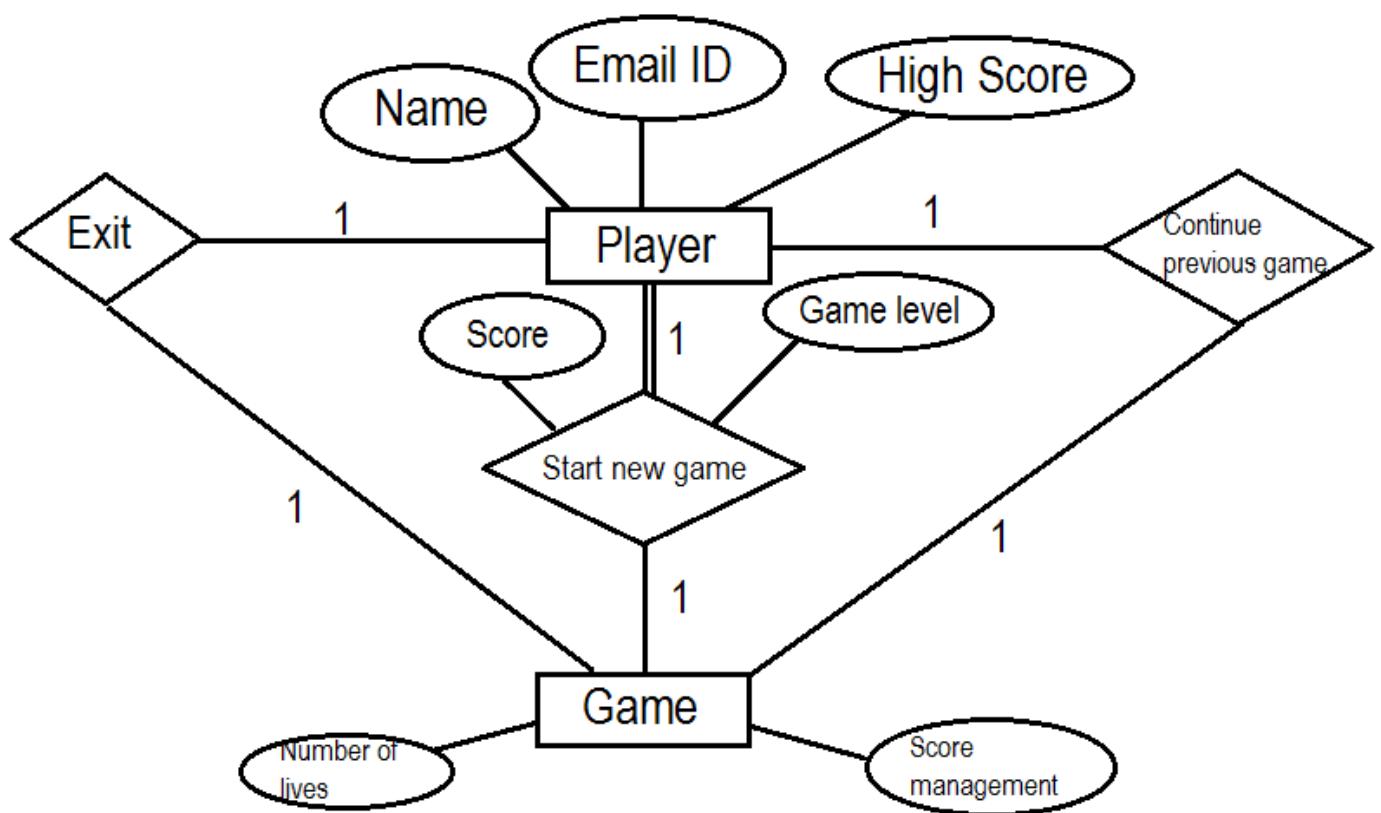
We thank **Tanay PrabhuDesai** for helping us get started on the right foot and showing us how to follow a consistent pattern in development and testing.

We thank **Saurabh Nanda** for taking the time off from his busy schedule to advise us, and for the trigonometry which inevitably made us question our sanity.

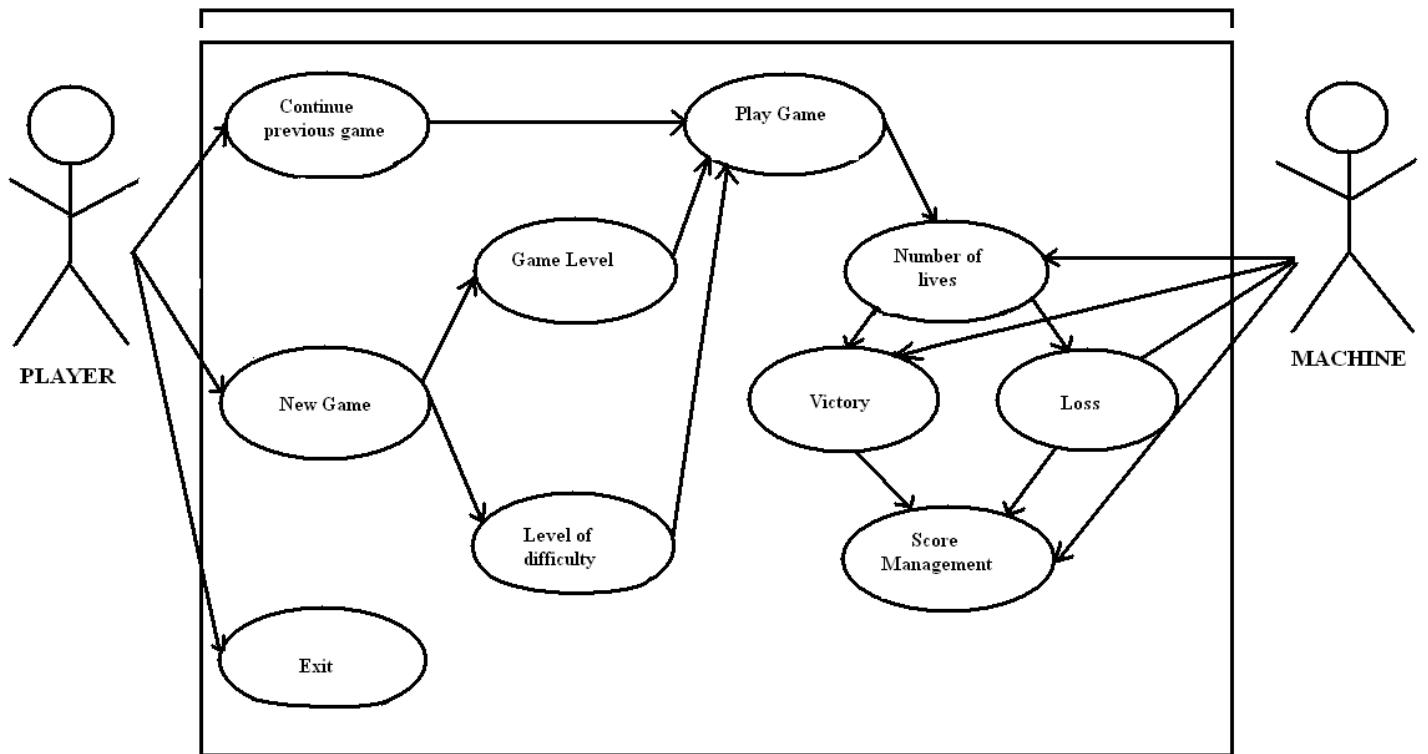
Any acknowledgement would be incomplete without thanking our parents for all the support they extended.

# Figures

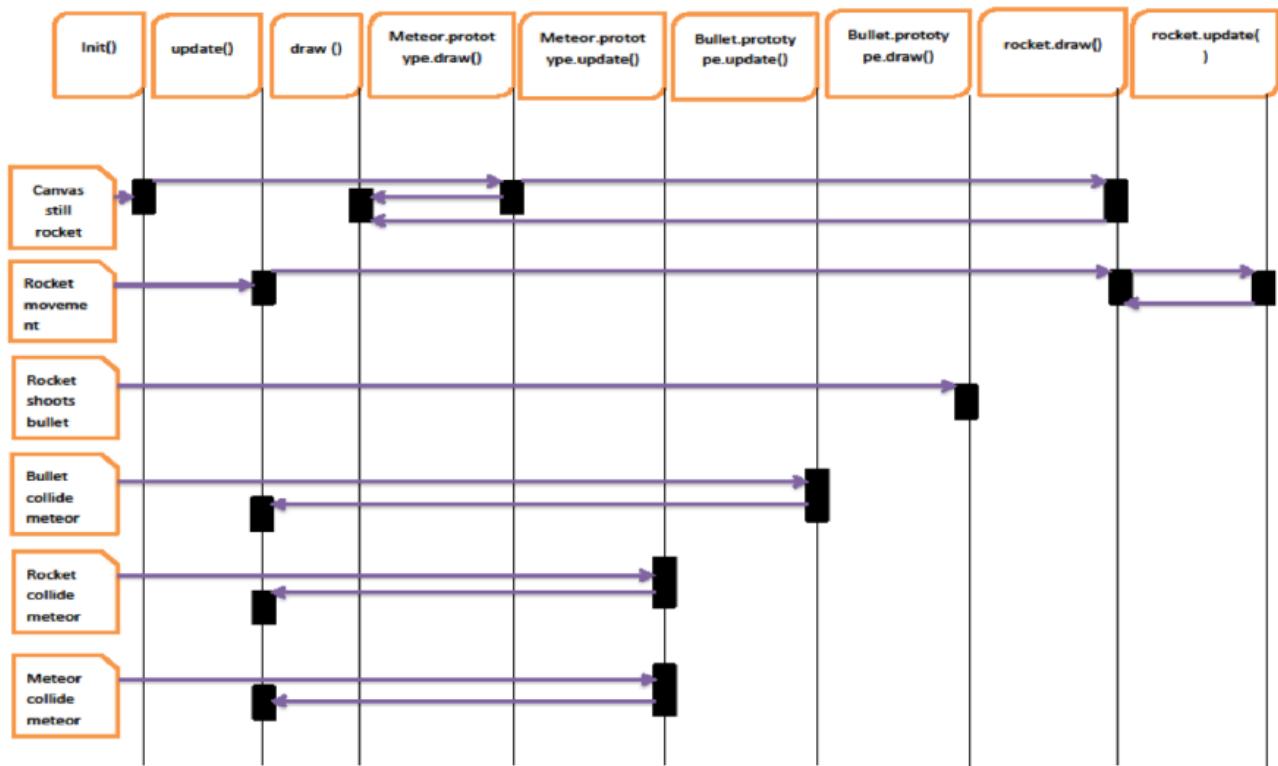
## 1. Entity-Relationship Diagram



## 2. Use Case Diagram



### 3. Sequence Diagram



# **1. Introduction**

Everybody likes to play games. The game we developed aims to be very light-weight and intuitive. There are no memory leaks, nor CPU intensive tasks.

It consists of a space environment with freely moving meteors. The user controls a rocket and shoots at the meteors to break them up. The player is provided with 3 lives at the beginning of the game.

At the end of the game the user enters his name and has the option to share his score with friends.

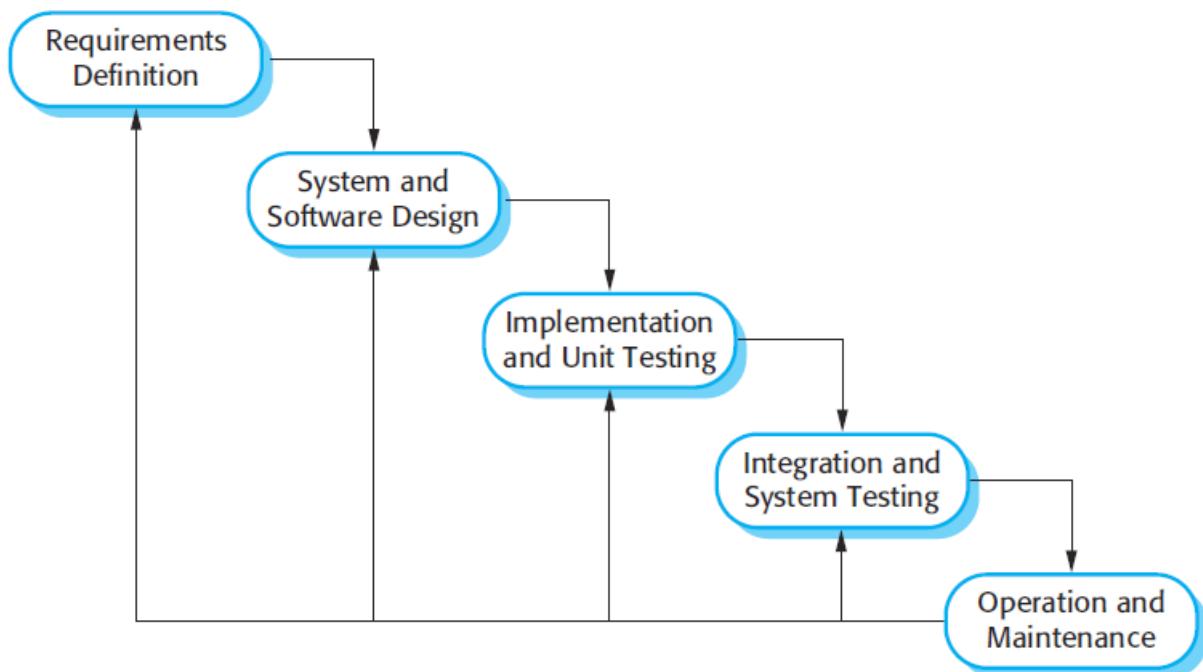
No external game engine was used in the creation of Asteroid.

The entire game logic is written in JavaScript and HTML5 & CSS3 are used for the front end. Ruby's Sinatra is used on the back end. The game does not require high bandwidth and loads minified files to improve already fast load times.

The biggest challenge proved to be the calculations for speed and collision detection.

## 2. Project Organisation

### 2.1 Process Model



### 2.2 Tools and Techniques

Development Environment:

- **Operating system:** Ubuntu 14.04 LTS
- **CPU:** Intel x86/x86\_64 architecture CPU
- **Graphics:** NVIDIA GeForce 710M
- **Editor:** Sublime Text 3 (beta)
- Ruby-2.2.1
- Sinatra 1.4.7
- JavaScript (ECMAScript 6)
- SQLite3 & PostgreSQL
- Git 1.9.1
- Gimp 2.8.10

## **3. Software Requirements Specification**

### **3.1 Purpose**

The purpose of this document is to give a detailed description of the requirements for the Asteroid. It will illustrate the purpose and complete declaration for the development of the system. It will also explain system constraints, interface and interactions with other external applications.

### **3.2 Document Conventions**

#### **Main Sections Titles:**

Font: Liberation Serif  
Face: Bold  
Size: 20

#### **Subsection Titles:**

Font: Liberation Serif  
Face: Bold  
Size: 18

#### **Other Text Explanations:**

Font: Liberation Serif  
Face: Normal  
Size: 14

### **3.3 Intended Audience and Reading Suggestions**

This Software Requirements document is intended for:

- **Developers** who can review project's capabilities and more easily understand where their efforts should be targeted to improve or add more features to it (design and code the application – it sets the guidelines for future development).
- **Project testers** can use this document as a base for their testing strategy as some bugs are easier to find using a requirements document. This way testing becomes more methodically organized.
- **End users** of this application who wish to read about what this project can do.

## **3.4 Product Scope**

Asteroid is a game application which provides end users an interface to play a space based game.

It is a low bandwidth game that is fun and intuitive, with a lot of visual cues.

We hope to foster a competitive gaming environment.

## 4. Overall Description

### 4.1 Directory structure

```
.  
|-- public  
|   |-- audio  
|   |-- css  
|   |-- images  
|   |   '-- originals  
|   '-- js  
|-- reviews  
|   '-- 7th sem review  
|       |-- images  
|       '-- SRS  
 '-- views
```

### 4.2 Product Functions

- Meteor movement
  - Meteor.update : updates meteor coordinates
  - Meteor.draw : draws meteor on canvas
- Rocket movement
  - Rocket.update : updates rocket coordinates
  - Rocket.draw : draws rocket on canvas
- Bullet movement
  - Bullet.update : updates bullet coordinates
  - Bullet.draw : draws bullet on canvas
- Wrap around - Wraps around meteors and rocket

```
if (this.x > global.width) {  
    this.x = 0;  
} else if (this.x < 0) {  
    this.x = global.width;  
} else if (this.y > global.height) {  
    this.y = 0;  
} else if (this.y < 0) {  
    this.y = global.height;  
};
```
- Collision detection
  - Rocket & Meteors
  - Bullet & Meteors
  - Meteors
- Game audio - Audio for the game ( crash, explosions)
- Lives
- User Scoreboard - List of all scores

## 4.3 System interfaces

Asteroid integrates two internal systems to provide functionality:

- **Client** The game has an interface to the user's client to receive user input and moves selections for the game.
- **Network** The game has an interface to the network in order to transmit information.

## 4.4 Operating Environment

The game needs an Internet browser that supports HTML5 and JavaScript to run, preferably Google Chrome.

V8 is Google's open source high-performance JavaScript engine, written in C++ and used in Google Chrome. It ensures the JavaScript game engine runs at optimal speeds.

## **5. External Interface Requirements**

### **5.1 User Characteristics**

Asteroid is a very intuitive game and the only thing the user is required to know are the rules of the game, which are provided.

### **5.1 Hardware Interfaces**

Asteroid runs on any computer which meets the following criteria:

- Capable to use an Internet connection
- Includes memory storage
- Includes a mouse
- Includes a keyboard

### **5.3 Software Interfaces**

Requires an HTML5 and JavaScript enabled web browser.

## 6. Other Nonfunctional Requirements

### 6.1 Performance Requirements

The game is expected not to hang nor lag. The collision detection should be immediate.

### 6.2 Software Quality Attributes

The game should be

- **Correct:** It should ‘just work’.
- **Flexible:** The user should be able to move, shoot, and carry out operations however he wants to. There will be no fixed movements.
- **Maintainable:** The code should be clean, modular, and adhere to standards.
- **Portable:** The game engine should be portable to any framework.
- **Reliable:** The physics behind the game must be reliable and work predictably. There should be no undefined results.
- **Testable:** The game engine should be easy to test; the code should provide for easy testing, deleting, etc.

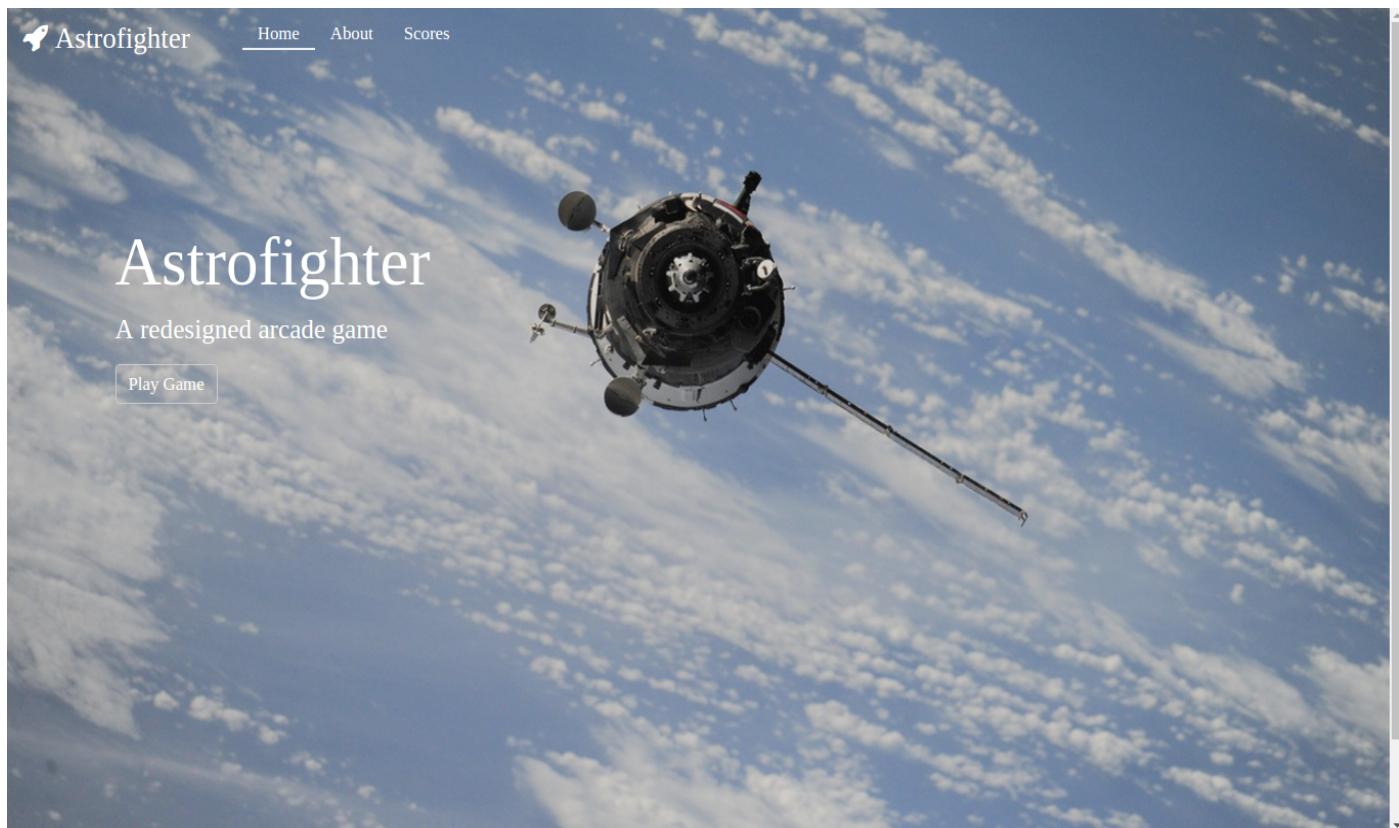
## 7. Implementation Details

```
asteroid/
|-- config.ru
|-- development.db
|-- Gemfile
|-- Gemfile.lock
|-- LICENSE
|-- main.rb
|-- player.rb
|-- populate_db.rb
|-- public/
|   |-- audio/
|   |-- css/
|   |-- images/
|   |   '-- originals/
|   '-- js/
|-- README.md
|-- tasks.todo
"-- views/
```

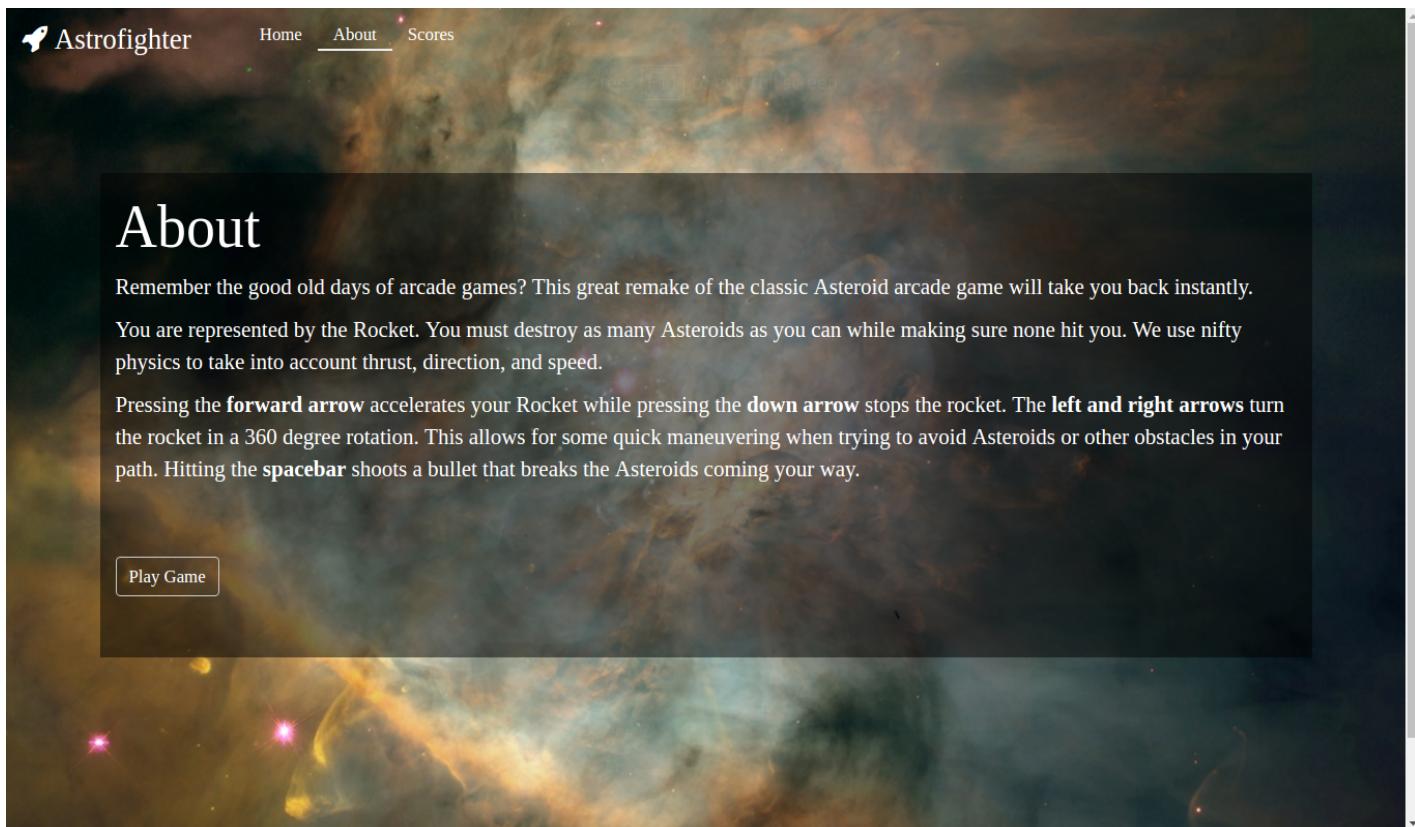
### 7.1 Routes:

The game consists of various routes, each of which has a specific function.

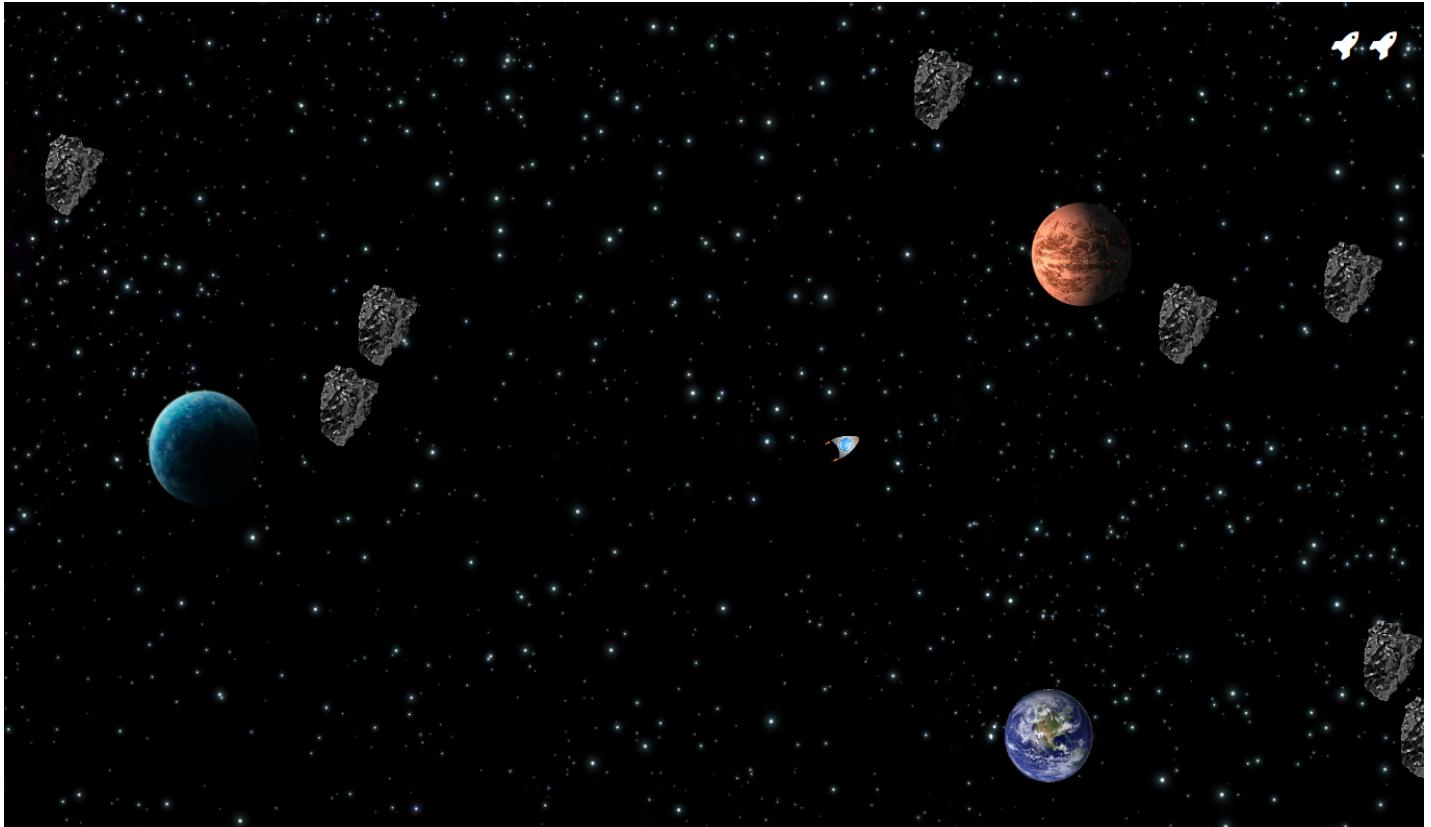
- GET / - home page



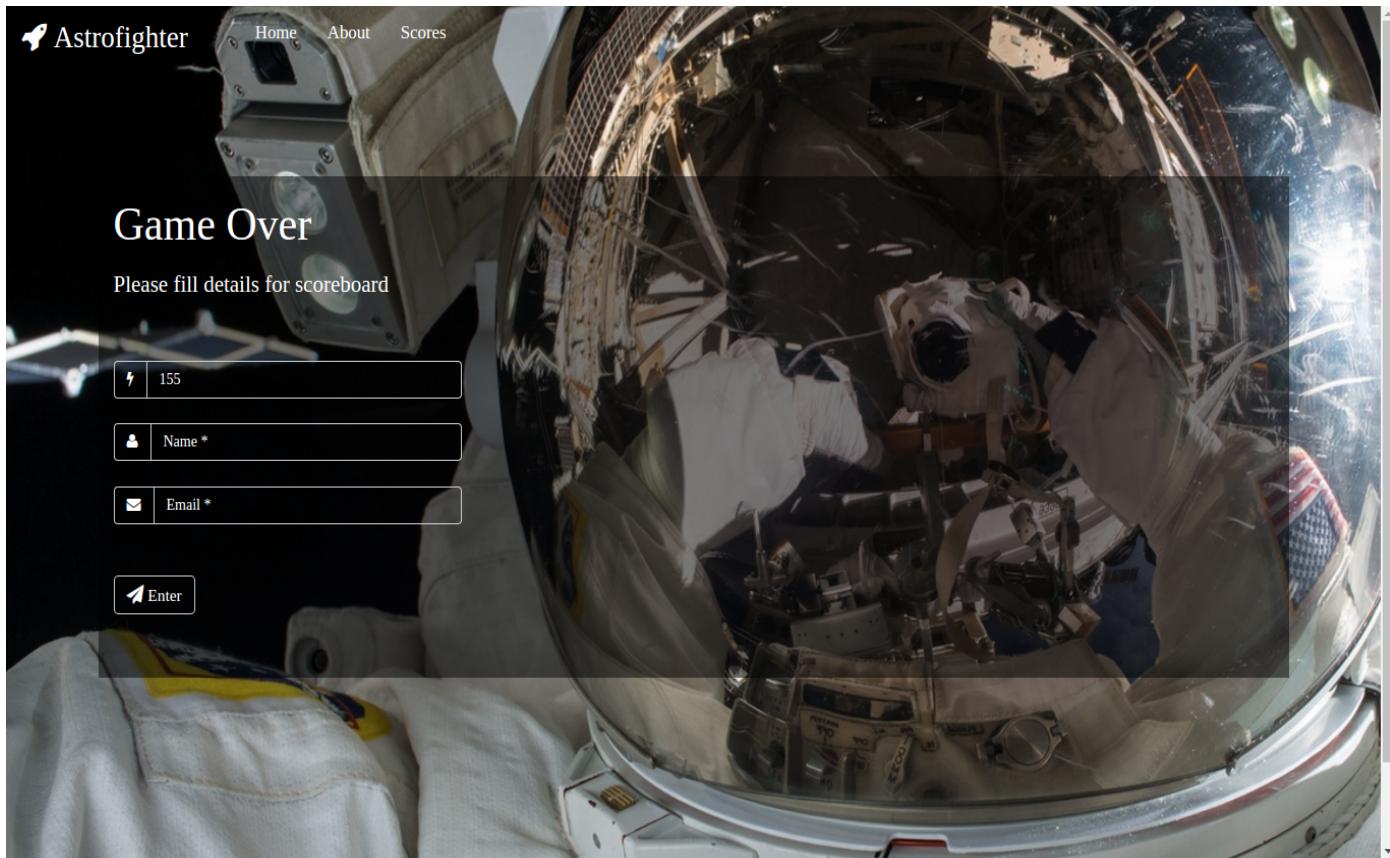
- GET /about – About page



- GET /game/? - Game page



- GET /gameover/? - Game-over page. Enter use details



- GET /scores - Scoreboard page

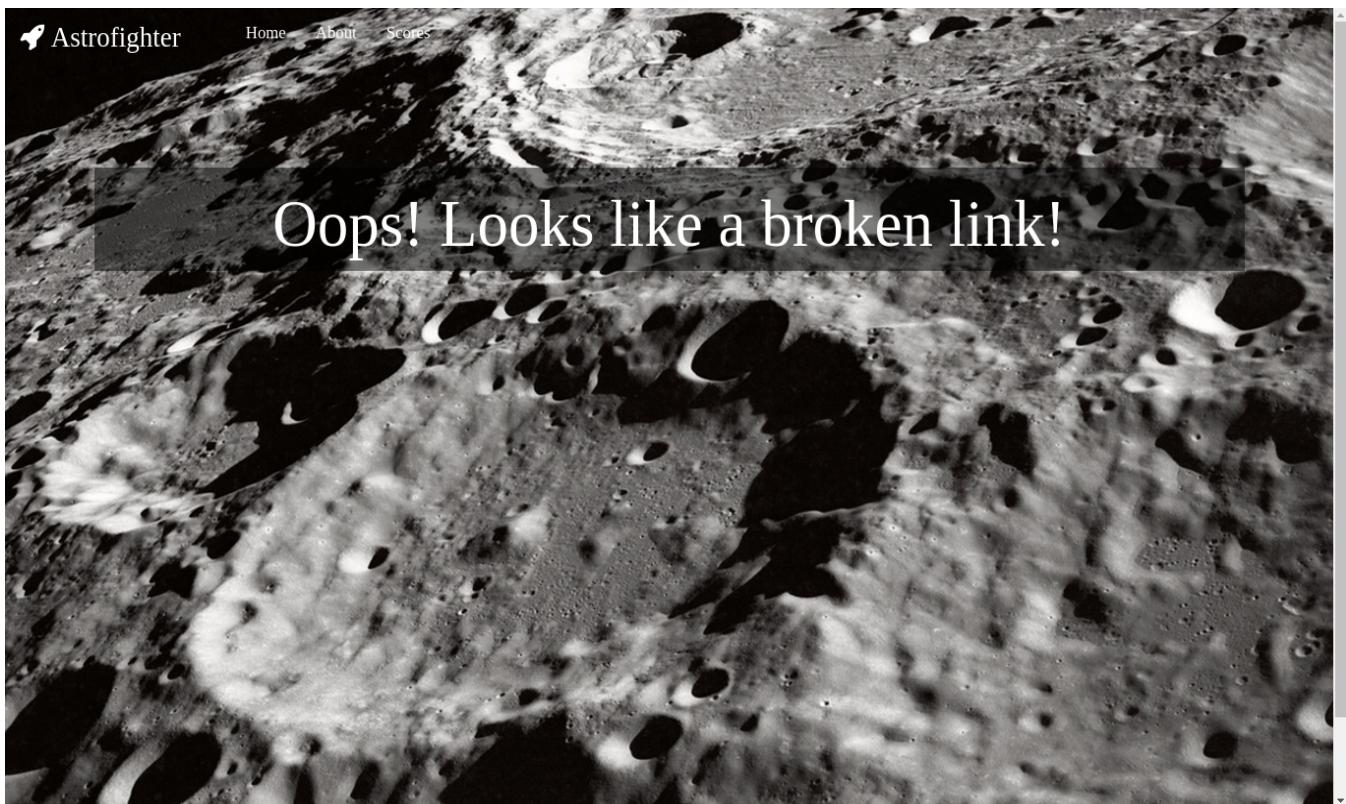
Astrofighter

Home About Scores

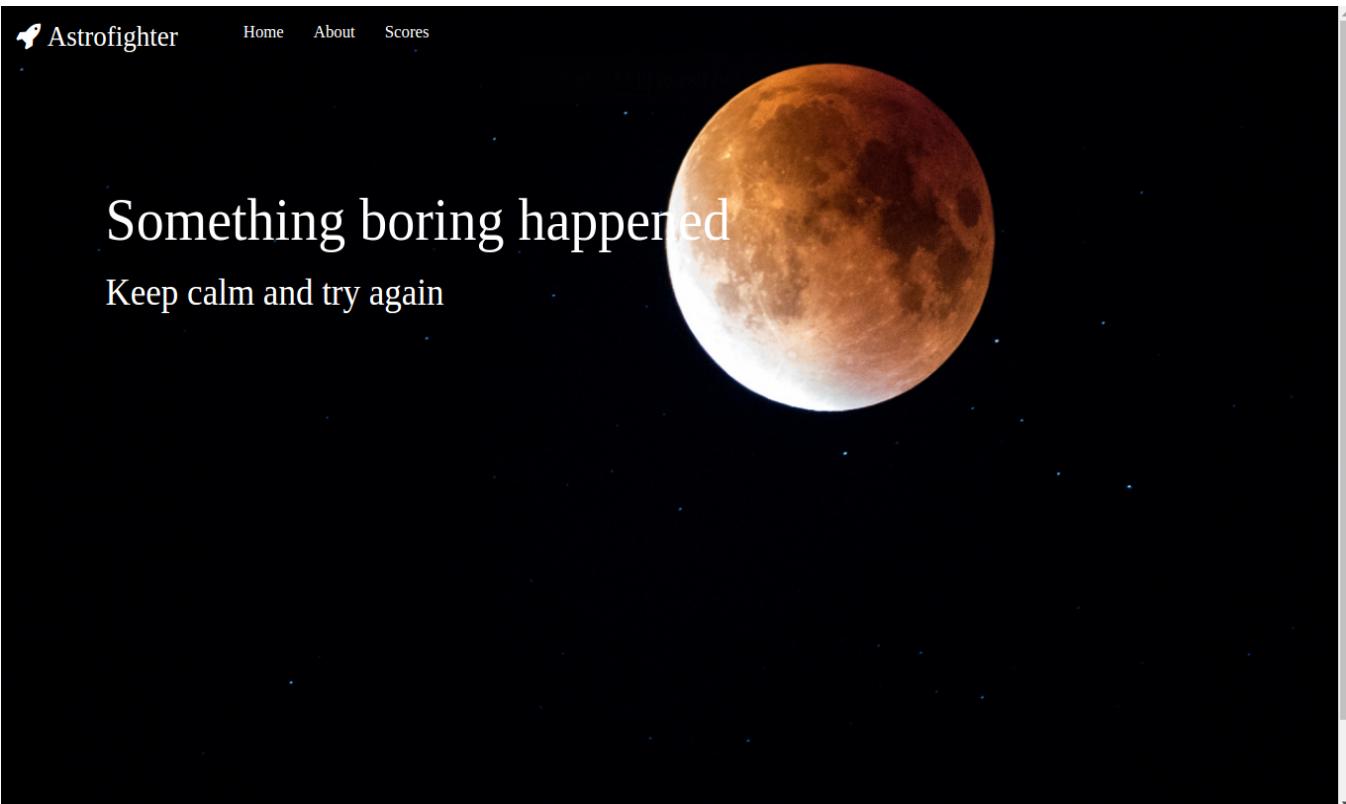
## Top space explorers

aaron	775
user3	19
user0	18
user1	11
user4	11
user5	10
user9	6
user7	2
user2	1
user6	1
user8	0

- POST /players - POST request route to update/add Player details
- **Not Found Page** – Broken Links



- **Server Error Page** – Unknown error on server side



## 7.2 Views

ERB (Embedded Ruby) is a feature of Ruby that enables you to conveniently generate any kind of text, in any quantity, from templates. The templates themselves combine plain text with Ruby code for variable substitution and flow control, which makes them easy to write and maintain.

ERB was used to generate the views for each of the routes.

```
views/
|-- about.erb
|-- error.erb
|-- game.erb
|-- game_over.erb
|-- home.erb
|-- layout.erb
|-- layout_head.erb
|-- nav.erb
|-- not_found.erb
-- scores.erb
```

## 7.3 Styles

CSS3 is the latest evolution of the Cascading Style Sheets language.

The views are styled using CSS3. The files are:

```
public/css/
|-- about.css
|-- error.css
|-- game_over.css
|-- home.css
|-- layout.css
|-- main.css
|-- not_found.css
-- scores.css
```

## 7.4 Player Class:

The player class is defined as:

```
class Player
  include DataMapper::Resource
  property :id, Serial
  property :email, String
  property :name, String
  property :score, Integer
end
```

## 7.5 Audio

Audio files are stored in </public/audio/>

## 7.6 Images

Image files are stored in </public/images/>

## 7.7 Game

### 7.7.1 Objects

- Rocket Object

```
var rocket = {
    spriteNormal: null,
    spriteMoving: null,
    x: 0,
    y: 0,
    width: 20,
    height: 30,
    speed: 0,
    angle: 90,
    move: false,
    accelerate: false,
    rotating: false,
    clkwise: true,
    draw : null,
    up: null,
    down: null,
    left: null,
    right: null,
    update: null
};
```

- Asteroid Object

```
var Meteor = function(meteorSize) {
    this.sprite = document.getElementById('asteroid');
    this.x = Math.random() * global.width;
    this.y = Math.random() * global.height;
    this.width = meteorSize;
    this.height = meteorSize;
    this.speed = 0;
    this.xmovement = Math.random() * 2;
    this.ymovement = Math.random() * 2;
    this.angle = 30;
    this.rotating = false;
    this.clkwise = true;
};
```

- Bullet Object

```
var Bullet = function() {  
    this.sprite = document.getElementById('bullet');  
    this.x = rocket.x + 1 ;  
    this.y = rocket.y + 6.5 ;  
    this.width = 5 ;  
    this.height = 5 ;  
    this.speed = 7 ;  
    this.angle = rocket.angle;  
    this.hit = false;  
};
```

## 7.7.2 Functions

- Initialization Function

First function that is called. It initializes all the data we need.

```
function init () {
    canvas = document.getElementById('game-canvas');
    context = canvas.getContext('2d');
    global.width = canvas.width = window.innerWidth;
    global.height = canvas.height = window.innerHeight;
    global.left = 37, global.up = 38, global.right = 39, global.down = 40;
    global.space = 32;

    rocket.x = (rocket.width + global.width)/2;
    rocket.y = (rocket.height + global.height)/2;

    rocket.spriteNormal = document.getElementById('rocket-normal');
    rocket.spriteMoving = document.getElementById('rocket-moving');

    for (let i = 0; i < global.initialMeteorNumber; i++) {
        meteor.push(new Meteor(global.meteorSizeLarge));
    };

    document.addEventListener('keydown', keydown);
    document.addEventListener('keyup', keyup);

    console.log( "Lives left: " + global.lives);
    console.log( "Score: " + global.score);
};
```

- Step Function

It is called 65 times a second by `requestAnimationFrame`. It calls global update and draw functions followed by a recursive call to itself. This form of recursion is called tail call optimization and ensures linear complexity without a stack.

```
var step = function (timestamp) {
    update();
    draw();
    window.requestAnimationFrame(step);
};
window.requestAnimationFrame(step);
```

- Global Update Function

Calls the update function of each object.

```
function update () {
    rocket.update();

    for (let met of meteor) {
        met.update();
    };

    for(let shot of bullet) {
        shot.update();
    };
}
```

- Global Draw Function

Calls the draw function of each object

```
function draw () {
    // clear full context to redraw new stuff
    context.clearRect(0, 0, global.width, global.height);
    rocket.draw();

    for (let met of meteor) {
        met.draw();
    };

    for (let shot of bullet) {
        shot.draw()
    };
}
```

- Meteor – Draw & Update Functions

```
Meteor.prototype.draw = function() {
    context.drawImage(this.sprite, this.x, this.y, this.width, this.height);
};

Meteor.prototype.update = function() {

    this.x += this.xmovement;
    this.y += this.ymovement;
    //to wrap the asteroids
    if (this.x > global.width) {
        this.x = 0;
    } else if (this.x < 0) {
        this.x = global.width;
    } else if (this.y > global.height) {
        this.y = 0;
    } else if (this.y < 0) {
        this.y = global.height;
    };

    /*Collision detection with rocket*/
    if ( (Math.abs(this.x - rocket.x) < 30) && (Math.abs(this.y - rocket.y) < 30) ) {
        rocketCrash();
    };
};
```

- Bullet – Draw & Update Functions (folded code)

```
Bullet.prototype.draw = function() {
    context.drawImage(this.sprite, this.x, this.y, this.width, this.height);
};

Bullet.prototype.update = function() {

    this.x += Math.sin(this.angle*(Math.PI/180))*this.speed;
    this.y += Math.cos(this.angle*(Math.PI/180))*this.speed*-1;

    /*Collision detection between bullet & meteor*/
    for (let met of meteor) {
        /*if 'in' is used, met => index no. With 'of', met => values(object)*/
        if ( (Math.abs(this.x - met.x) < 25) && (Math.abs(this.y - met.y) < 25) && (this.hit === false) ) {
            explosionAudio.play(); // meteor break-up audio

            // Get hit meteor's metadata
            let meteorSize = met.width;

            // Bullet & Meteor vanish
            this.sprite = document.getElementById('null');
            met.sprite = document.getElementById('null');

            // Remove bullet & meteor from their arrays
            bullet.splice(bullet.indexOf(this), 1);
            meteor.splice(meteor.indexOf(met), 1);

            // Break Meteor based on size
            switch(meteorSize) { ... }

            function addMeteors (newMeteorSize) { ... }
            // Making sure one bullet's hit is registered only once
            this.hit = true;
        };
    };
};
```

- Rocket- Draw Function

```
rocket.draw = function() {
    context.translate(rocket.x, rocket.y);
    context.rotate(rocket.angle*(Math.PI/180)); /*Rotate context to angle required*/

    if (rocket.move) {
        context.drawImage(rocket.spriteMoving, 0, 0, rocket.width, rocket.height+4);
    } else {
        context.drawImage(rocket.spriteNormal, 0, 0, rocket.width, rocket.height);
    };

    context.rotate(rocket.angle*(Math.PI/180) * -1);
    context.translate(-rocket.x, -rocket.y);
};
```

- Rocket – Update Function

```
rocket.update = function() {

    if (rocket.move) {
        rocket.x += Math.sin(rocket.angle*(Math.PI/180))*rocket.speed;
        rocket.y += Math.cos(rocket.angle*(Math.PI/180))*rocket.speed*-1;

        //place outside this 'if' to make it turn even when not moving
        if (rocket.rotating) {
            if (rocket.clkwise) {
                rocket.angle += 5;
            } else {
                rocket.angle -= 5;
            }
        };
    } else {
        rocket.speed = 0;
    };

    if (rocket.accelerate) {
        rocket.speed = 5;
    } else {
        rocket.speed *= friction;
    };

    /*Due to friction speed drops. When it goes below treshold, move = false
     so that sprite image is changed*/
    if (rocket.speed <= 0.3) {
        rocket.move = false;
    };

    //rocket wrap around
    if (rocket.x > global.width) {
        rocket.x = 0;
    } else if (rocket.x < 0) {
        rocket.x = global.width;
    } else if (rocket.y > global.height) {
        rocket.y = 0;
    } else if (rocket.y < 0) {
        rocket.y = global.height;
    };
};
```

- Key Listener Functions

Listen for key presses to perform actions.

```
functionkeydown(event) {  
    let key = event.keyCode;  
  
    switch(key) {  
        case global.left:  
            rocket.left('pressed');  
            break;  
        case global.up:  
            rocket.up('pressed');  
            break;  
        case global.right:  
            rocket.right('pressed');  
            break;  
        case global.down:  
            rocket.down();  
            break;  
        case global.space:  
            shotAudio.play(); // bullet fired audio  
            bullet.push(new Bullet());  
            break;  
    }  
};  
  
functionkeyup(event) {  
    let key = event.keyCode;  
  
    switch(key) {  
        case global.left:  
            rocket.left('released');  
            break;  
        case global.up:  
            rocket.up('released');  
            break;  
        case global.right:  
            rocket.right('released');  
            break;  
    }  
};
```

### 7.7.3 Global Data

```
var context = null, canvas, global = {};
var friction = 0.95;
var meteor = [], bullet = [];
var explosionAudio = new Audio('/audio/explosion.mp3');
var shotAudio = new Audio('/audio/shot.mp3');

global.initialMeteorNumber = 8;
global.meteorSizeLarge = 110;
global.meteorSizeMedium = 80;
global.meteorSizeSmall = 50;
// Points for breaking meteors
global.smallMeteorPoints = 5;
global.mediumMeteorPoints = 10;
global.largeMeteorPoints = 15;

global.lives = document.getElementById('lives').innerHTML;
global.score = parseInt(document.getElementById('score').innerHTML);
```

## 7.7.4 Integrated Code

All the functions and objects integrate to form a single script.

\* The code has been folded to ensure brevity.

```
function loaded () {  
  
    "use strict"      // So 'let' and other stuff can be used  
  
    var context = null, canvas, global = {};  
    var friction = 0.95;  
    var meteor = [], bullet = [];  
    var explosionAudio = new Audio('/audio/explosion.mp3');  
    var shotAudio = new Audio('/audio/shot.mp3');  
  
    global.initialMeteorNumber = 8;  
    global.meteorSizeLarge = 110;  
    global.meteorSizeMedium = 80;  
    global.meteorSizeSmall = 50;  
    // Points for breaking meteors  
    global.smallMeteorPoints = 5;  
    global.mediumMeteorPoints = 10;  
    global.largeMeteorPoints = 15;  
  
    global.lives = document.getElementById('lives').innerHTML;  
    global.score = parseInt(document.getElementById('score').innerHTML);  
  
    var rocket = { };  
  
    var Meteor = function(meteorSize) { };  
  
    var Bullet = function() { };  
  
    Bullet.prototype.draw = function() {  
        context.drawImage(this.sprite, this.x, this.y, this.width, this.height);  
    };  
  
    Bullet.prototype.update = function() { };
```

```
Meteor.prototype.draw = function() {
    context.drawImage(this.sprite, this.x, this.y, this.width, this.height);
};

Meteor.prototype.update = function() { };

rocket.up = function(status) {
    if (status == "pressed") {
        rocket.move = true;
        rocket.accelerate = true;
    } else {
        rocket.accelerate = false;
    };
};

rocket.down = function() {
    rocket.accelerate = false;
    rocket.move = false;
};

rocket.left = function(status) {
    if (status == "pressed") {
        rocket.rotating = true;
        rocket.clkwise = false;
    } else{
        rocket.rotating = false;
    };
};

rocket.right = function(status) {
    if (status == "pressed") {
        rocket.rotating = true;
        rocket.clkwise = true;
    } else{
        rocket.rotating = false;
    };
};
```

```
rocket.draw = function() {...};
};

rocket.update = function() {...};

function keydown (event) {...};

function keyup (event) {...};

function init () {...};

function update () {...};

function draw () {...};

init();

var step = function (timestamp) {...};
window.requestAnimationFrame(step);

function rocketCrash () {
    if (global.lives > 1) {
        global.lives = global.lives - 1;
        window.location = "/game?lives=" + global.lives + "&score=" + global.score;
        throw new Error("Thrown to stop script from continuing executing");
    } else if (global.lives == 1 || global.lives < 1) {
        window.location = "/gameover?score=" + global.score;
        throw new Error("Thrown to stop script from continuing executing");
    };
};

window.onload = loaded;
```

## 7.8 Gemfile

A Gemfile is a file we create which is used for describing gem dependencies for Ruby programs. A gem is a collection of Ruby code that we can extract into a “collection” which we can call later.

```
source "https://rubygems.org"
ruby '2.2.1'
gem "sinatra"
gem "slim"
gem "sass"
gem "dm-core"
gem "dm-migrations"
gem "thin"
gem "sinatra-flash"

gem "pg", :group => :production
gem "dm-postgres-adapter", :group => :production

gem "dm-sqlite-adapter", :group => :development
```