CrayonPencil Documentation

Files Tree

```
├──assets
│   ├──images
│   │   ├──bg_tile.jpg
│   │   ├──enemy1_eraser.png
│   │   ├──loading_screen.png
│   │   ├──mainship_pencil.png
│   │   └──pencil_projectile1_tip.png
│   ├──movieclips
│   │   ├──eraser_explosion.png
│   │   └──eraser_explosion.json
│   └──music
│       ├──Fireproof_Babies_-_Lilly....so_far_away.mp3
│       ├──Fireproof_Babies_-_Lilly....so_far_away.mp3
│       └──tracks_license.txt
├──css
│   └──layout.css
├──js
│   ├──classes
│   │   ├──enemy.js
│   │   ├──PreloadAudio.js
│   │   └──projectile.js
│   ├──levels
│   │   ├──level1.js
│   │   ├──level2.js
│   │   └──loading.js
│   ├──libs
│   │   ├──jquery-1.9.0
│   │   │   └──jquery.min.js
│   │   └──pixi.js
│   └──PositionCanvas.js
└──index.html
```

Game Logic

The game consists of 2 levels with different difficulties

The goal of the game

Is to get the score required to pass the level

- Level1**:** 2000 points
- Level2: 5000 points

Successful shot

Increase your score by 15 points

Score reset

When your pencil collide with enemy(eraser), you lose your score and the game reset again

level2 Differences

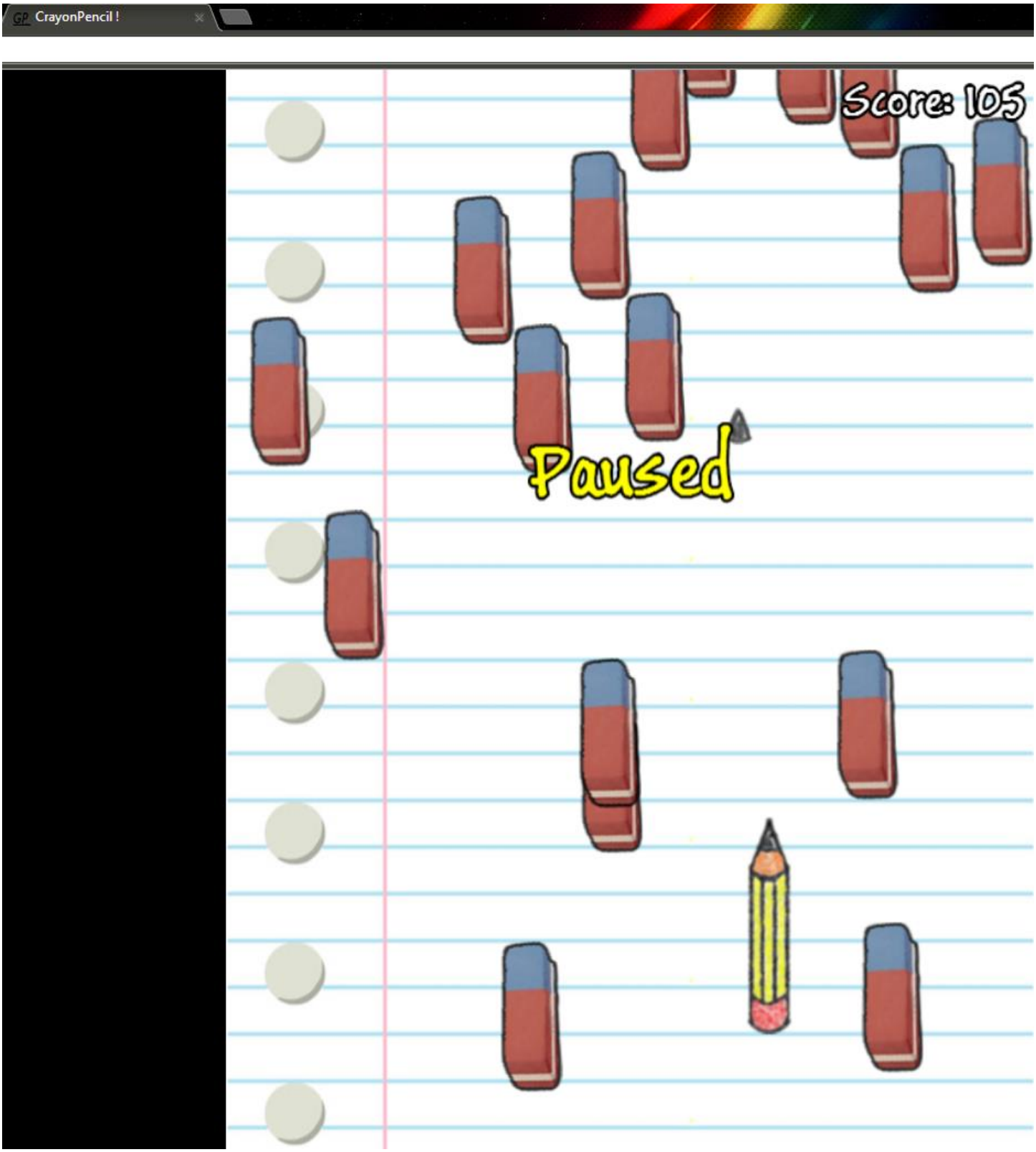
- Enemy moves in cosine wave form
- Frequency of creation
- Enemy speed
- Shooting frequency

Used Technologies

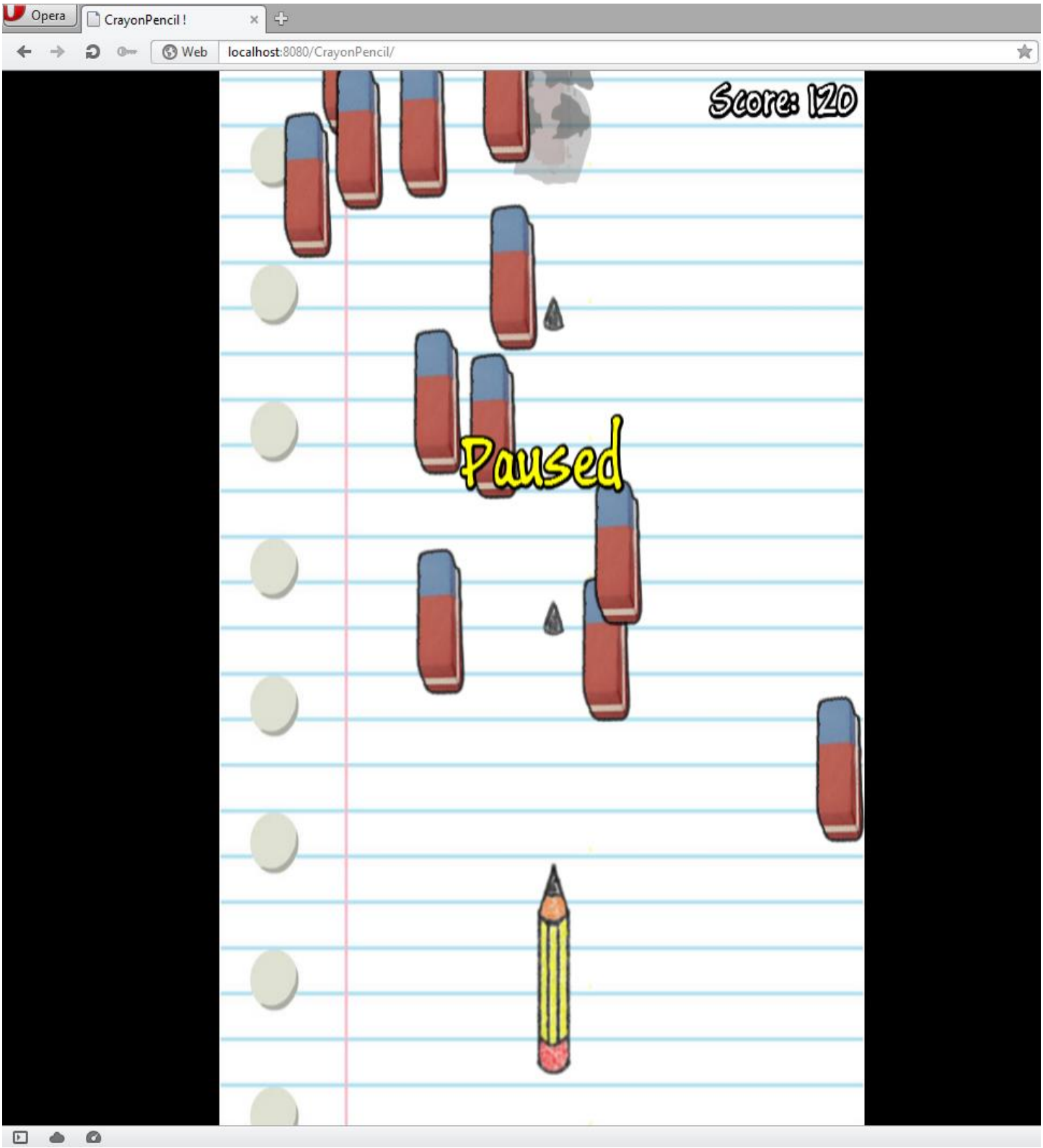
- HTML5
- Pixi rendering engine
- Jquery

Game Screen Shots

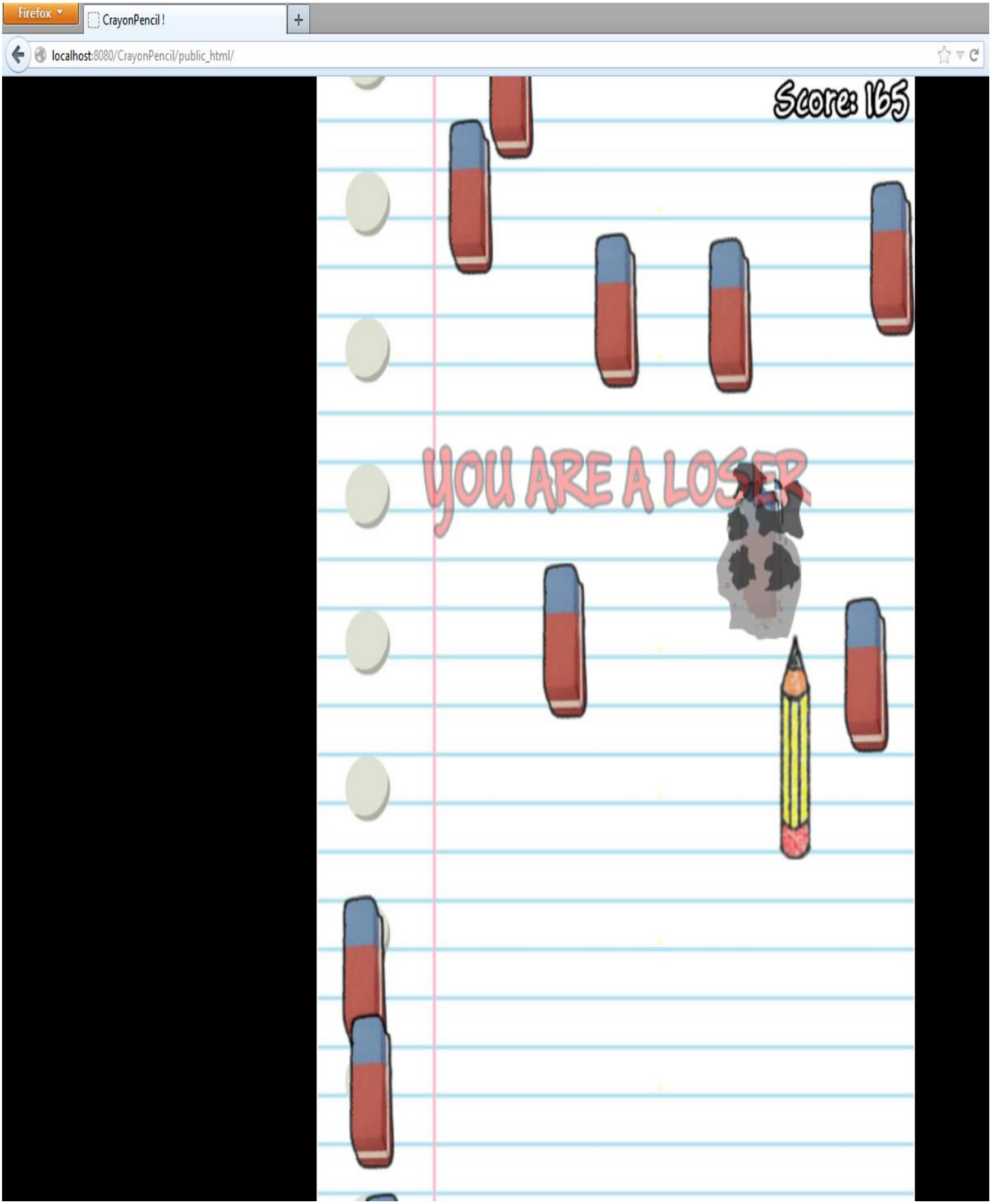
Chrome



Opera


Opera
CrayonPencil !
Web
localhost:8080/CrayonPencil/
Score: 120
Paused

FireFox


Firefox
CrayonPencil !
localhost:8080/CrayonPencil/public_html/
Score: 165
YOU ARE A LOSER

Code Snippets with description

PositionCanvas.js

→This file call **FixAppPos()** function that being used in centering the application canvas.

→SetInterval call **FixAppPos()** every 50 msec to keep the canvas position updated if any changes happened to the width or the height of the window.

```
jQuery(function()
{
    FixAppPos();
    setInterval(FixAppPos, 50);
});
```

Projectile.js

It's a class for the projectiles objects and its constructor take to prams (frameName, speed)

Class Variables

→sprite.anchor to change the origin of the object

```
this.frameName = frameName;
this.speed = speed;
this.sprite = PIXI.Sprite.fromFrame(frameName);
this.sprite.anchor.x = 0.5;
this.sprite.anchor.y = 0.5;
```

Class Functions

→**addToStage:** add the object to the stage

```
this.addToStage = function()
{
    stage.addChildAt(this.sprite,nTiles);
};
```

→**removeFromStage:** delete the object from the stage

```
this.removeFromStage = function()
{
    stage.removeChild(this.sprite);
};
```

→**animate:** move the projectile with delta y = to its speed

```
this.animate = function()
{
    this.sprite.position.y += speed;
};
```

→**isOutOfScreen:** return true if the object out of the screen

```
this.isOutOfScreen = function()
{
   return  (this.sprite.position.y < -this.sprite.height / 2) || (this.sprite.position.y > canvasHeight + this.sprite.height / 2);
};
```

→**setPosition:** used to initialize the position of the object

```
this.setPosition = function(x,y)
{
    this.sprite.position.x = x;
    this.sprite.position.y = y;
};
```

enemy.js

It's a class for the enemy objects and its constructor take prams (frameName, initialSpeed, speedRange)

Class Variables

→Enemy object states enumeration which determine the current state of the enemy (moving, exploding, exploded)

→sprite.anchor to change the origin of the object

→Speed range is a constant value multiplied by a fraction from 0 to 1 to make the speed of each instance of this class different from the others

```
var obj = this;
this.States = {
    Moving: 'moving',
    Exploding: 'exploding',
    Exploded: 'exploded'
};
this.state = this.States.Moving;
this.frameName = frameName;
this.speed = initialSpeed + Math.random() * speedRange;
this.speedRange = speedRange;
this.sprite = PIXI.Sprite.fromFrame(frameName);
this.sprite.anchor.x = 0.5;
this.sprite.anchor.y = 0.5;
this.explosion = new PIXI.MovieClip(explosionTextures);
this.explosion.anchor.x = 0.5;
this.explosion.anchor.y = 0.5;
```

Class Functions

→**addToStage:** add the object to the stage

```
this.addToStage = function()
{
    stage.addChildAt(this.sprite, nTiles);
};
```

→**removeFromStage:** set the state of the object to exploded and delete the it from the stage

```
this.removeFromStage = function()
{
    this.state = this.States.Exploded;
    try
    {
        stage.removeChild(this.sprite);
    } catch (e) {
    }
    try
    {
        stage.removeChild(this.explosion);
    } catch (e) {
    }
};
```

→**blowUp:** set the state of the object to exploding, change the position of the explosion sprite to the same position of the enemy position and start the animation

```
this.blowUp = function()
{
    this.state = this.States.Exploding;
    this.explosion.position.x = this.sprite.position.x;
    this.explosion.position.y = this.sprite.position.y;
    this.explosion.loop = false;
    this.explosion.gotoAndPlay(0);
    stage.addChildAt(this.explosion, nTiles);
    stage.removeChild(this.sprite);
};
```

→**movement:** move the enemy with delta y = to its speed

```
this.movement = function()
{
    this.sprite.position.y += this.speed;
};
```

→**animate:**

if the object state is moving

call **movement** function

else if state is exploading

fade the explosion out and remove it from the stage

```
this.animate = function()
{
    switch (this.state)
    {
        case this.States.Moving:
            this.movement();
            break;
        case this.States.Exploding:
            this.explosion.alpha -= 0.03;
            if (!this.explosion.playing)
            {
                stage.removeChild(this.explosion);
                this.state = this.States.Exploded;
            }
    }
};
```

→**isOutOfScreen:** return true if the object out of the screen

```
this.isOutOfScreen = function()
{
    return  (this.sprite.position.y > canvasHeight + this.sprite.height / 2);
};
```

→**setPosition:** used to initialize the position of the object

```
this.setPosition = function(x, y)
{
    this.sprite.position.x = x;
    this.sprite.position.y = y;
};
```

→**placeAtStart:** select a random x position for the enemy and y position = - object height / 2

```
this.placeAtStart = function()
{
    var proposedRandPos = Math.random() * canvasWidth + this.sprite.width / 2;
    this.sprite.position.x = proposedRandPos > (canvasWidth - this.sprite.width / 2) ? (canvasWidth - this.sprite.width / 2) : proposedRandPos;
    this.sprite.position.y = 0 - this.sprite.height / 2;
};
```

PreloadAudio.html

It's a class to preload audio files and its constructor take to prams (array of audio files that you want to load)

Class Variables

```
this.filesArray = filesArray;
this.filesLeft = this.filesArray.length;
this.browserSupport = getAudioSupport();
this.audioArray = [];
var obj = this;
```

Class Functions

→**onProgress, onNoSupport, onComplete:** these functions print messages in console for debugging purposes

```
function onProgress()
{
    console.log('file loaded, ' + this.filesLeft + ' left');
};
this.onNoSupport = function()
{
    console.log('this broswer doesn\'t support audio tags please provide a handler for this function');
};
this.onComplete = function()
{
    console.log('finished loading all audio please support a handler for this function');
};
```

→**getAudioSupport:** returns the extension that the browser supports and returns none if all of the available audio files not supported

```
function getAudioSupport()
{
    var a = document.createElement('audio');
    var ogg = !!(a.canPlayType && a.canPlayType('audio/ogg; codecs="vorbis"').replace(/no/, ''));
    if (ogg)
        return 'ogg';
    var mp3 = !!(a.canPlayType && a.canPlayType('audio/mpeg;').replace(/no/, ''));
    if (mp3)
        return 'mp3';
    else
        return 'none';
}
```

→**load:** check the type of audio files that the browser support, load the audio and put it in array

```
this.load = function()
{
    if (this.browserSupport === 'none')
        this.onNoSupport();
    else
    {
        for (var i = 0; i < this.filesArray.length; i++)
        {
            var audio = new Audio();
            audio.src = this.filesArray[i] + '.' + this.browserSupport;
            audio.preload = "auto";
            $(audio).on("loadeddata", this.FileFinished);
            this.audioArray.push(audio);
        }
    }
};
```

→**FileFinished**

```
this.FileFinished = function()
{
    --obj.filesLeft;
    obj.onProgress();
    if (obj.filesLeft === 0)
        obj.onComplete();
};
```

loading.js

At the beginning, it load the elements used in the preloading screen

```
this.loadingLevelFrames = [bgTileFrameName, loading_screenFrameName, pencilFrameName];
this.loader = new PIXI.AssetLoader(this.loadingLevelFrames);
```

The start preloading of the game assets

level1.js & level2.js

→The same file structure with different values

Index.html

Global variables

→The width and the height of the game canvas

→Game states enumeration which determine the current state of the game

```
canvasWidth = 600;
canvasHeight = 800;

GameStates = {
    WaitingForState: 'waiting',
    Loading: 'loading',
    Menu: 'menu',
    Lost: 'lost',
    Level1: 'level1',
    Level2: 'level2'
};

gameState = GameStates.Loading;
```

→Call the loading level class to start the game loading

```
var Loader = new LoadingLevel(toLoad, preloadAudio);
Loader.FinishedLoading = FinishedLoading;
```

→**animate:** check the game state and depending on that chose the appropriate animation function

```
function animate()
{
    requestAnimFrame(animate);
    switch (gameState)
    {
        case GameStates.WaitingForState:
            break;
        case GameStates.Loading:
            Loader.animate();
            break;
        case GameStates.Level1:
            level1.animate();
            break;
        case GameStates.Level2:
            level2.animate();
            break;
        default:
            console.error('Unknown game state ' + gameState);
    }
    renderer.render(stage);
}
```