Coder son jeu de shoot 2D avec HTML5 et Javascript

Alan Menant

INTRO

http://github.com/ZenikaOuest/shootZup/

SOMMAIRE

01 ANIMATION GRAPHIQUE

02 GESTION DU SON

03 GESTION DES COMMANDES

04 AUTOMATISATION DES ENNEMIS

05 HITBOXES ET COLLISIONS

06 ORCHESTRATION DU JEU

01 ANIMATION GRAPHIQUE

01 ANIMATION GRAPHIQUE

Balise HTML5 <canvas>

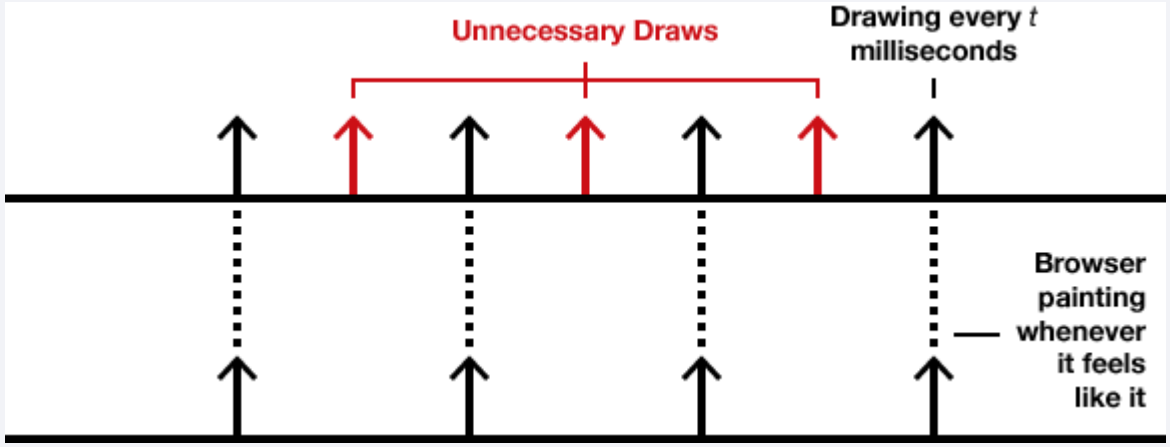
Accélération GPU

```
<!DOCTYPE html>
<html>

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-utf8">
    <title>Shoot'em up</title>
    <link rel="stylesheet" href="resources/game.css" />
</head>

<body>
    <canvas id="game" width="480" height="640"></canvas>

    <button id="mute" type="button" class="transparent-btn btn-mute-on"></button>

    <script src="audioManager.js"></script>
```

drawImage pour dessiner une image dans un canvas

01 ANIMATION GRAPHIQUE

Affichage des images @ 60 fps avec requestAnimationFrame




```
/**
 * Affichage des éléments du jeux
 */
Game.prototype.paint = function () {

    // TODO affichage du jeu

    window.requestAnimationFrame(this.paint.bind(this));
};
```

01 ANIMATION GRAPHIQUE

Changement des images pour donner l’illusion d’ animation

```
(function (window) {
    'use strict';

    function Sprite(x, y, frameSize, animations) {

        // Position de l'animation courante parmi les frames
        this.currentAnimationFrame = 0;

        // Taille en pixels d'une frame d'animation
        this.frameSize = frameSize;

        // Positions
        this.x = x;
        this.y = y;

        // Description de l'animation
        this.animations = animations;

        // Position y de l'animation courante dans le sprite général.
        this.animationY = 0;

        // Sert à controler la boucle d'animation
        this.animationLoopTimeoutId = null;
        this.animationLoopIntervalId = null;

        this.currentState = '';
    }
```

01 ANIMATION GRAPHIQUE

Animation en boucle

```
41      Sprite.prototype.startLoop = function (animationName) {
42          return new Promise(function (resolve) {
43
44              if (this.currentState !== animationName) {
45                  this.clearCurrentAnimation();
46
47                  var animation = this.animations[animationName];
48                  this.animationY = animation.animationY;
49                  // FIXME utile ?
50                  this.frameSize.width = animation.animationFrameWidth;
51
52                  this.currentState = animationName;
53
54                  this.animationLoopIntervalId = setInterval(function () {
55                      this.animLoop(animationName);
56                  }.bind(this), animation.speedRate);
57                  resolve();
58              }
59
60          }.bind(this));
61      };
62
63      Sprite.prototype.animLoop = function (animationName) {
64          this.currentAnimationFrame += 1;
65          if (this.currentAnimationFrame === this.animations[animationName].nbFrames) {
66              this.currentAnimationFrame = 0;
67          }
68      };
```

01 ANIMATION GRAPHIQUE

Animation une seule fois

```
71      Sprite.prototype.startOnce = function (animationName) {
72          return new Promise(function (resolve) {
73
74              if (this.currentState !== animationName) {
75                  this.clearCurrentAnimation();
76                  this.currentState = animationName;
77
78                  this.animationLoopTimeoutId = setTimeout(function () {
79                      this.animOnce(animationName).then(function () {
80                          resolve();
81                      });
82                  }.bind(this), this.animations[animationName].speedRate);
83              }
84
85          }.bind(this));
86      };
87
88      Sprite.prototype.animOnce = function (animationName) {
89          return new Promise(function (resolve) {
90
91              this.currentAnimationFrame += 1;
92              if (this.currentAnimationFrame !== this.animations[animationName].nbFrames) {
93                  this.animationLoopTimeoutId = setTimeout(function () {
94                      this.animOnce(animationName).then(function () {
95                          resolve();
96                      });
97                  }.bind(this), this.animations[animationName].speedRate);
98              } else {
99                  resolve();
100             }
101         }.bind(this));
```

01 ANIMATION GRAPHIQUE

Prototype générique pour un vaisseau

```
(function () {
    'use strict';

    function Spaceship(x, y, imageName, frameSize, hitboxSize, fly, resources) {
        this.resources = resources;
        this.imageName = imageName;

        // Taille en pixels de la hitbox
        this.hitboxSize = hitboxSize;

        var animations = {
            'FLY': fly
        };

        Sprite.call(this, x, y, frameSize, animations);
    }

    Spaceship.prototype = Object.create(Sprite.prototype);

    Spaceship.prototype.FLY = 'FLY';

    Spaceship.prototype.paint = function (context) {

        if (this.currentState === this.FLY) {
            context.drawImage(this.resources.images[this.imageName],
                this.currentAnimationFrame * this.frameSize.width, this.animationY,
                this.frameSize.width, this.frameSize.height,
                // centrage de l'image par rapport à la position
                this.x - (this.frameSize.width / 2), this.y - this.frameSize.height + (this.frameSize.height / 2),
                this.frameSize.width, this.frameSize.height
            );
        }
    };
```

01 ANIMATION GRAPHIQUE

Prototype pour le vaisseau bleu

```
(function (window) {
    'use strict';

    function SpaceshipBlue(x, y, resources) {

        // Taille en pixels d'une frame d'animation
        var frameSize = {
                width: 55,
                height: 60
            },
            // Taille en pixels de la hitbox
            hitboxSize = {
                width: 40,
                height: 45
            },
            // paramètre de l'animation
            fly = {
                nbFrames: 12,
                animationFrameWidth: frameSize.width,
                animationY: 0,
                speedRate: 60
            };

        Spaceship.call(this, x, y, 'spaceship-blue', frameSize, hitboxSize, fly, resources);
    }

    SpaceshipBlue.prototype = Object.create(Spaceship.prototype);

    window.SpaceshipBlue = SpaceshipBlue;

})(window);
```

02 GESTION DU SON

02 GESTION DU SON

Web Audio API

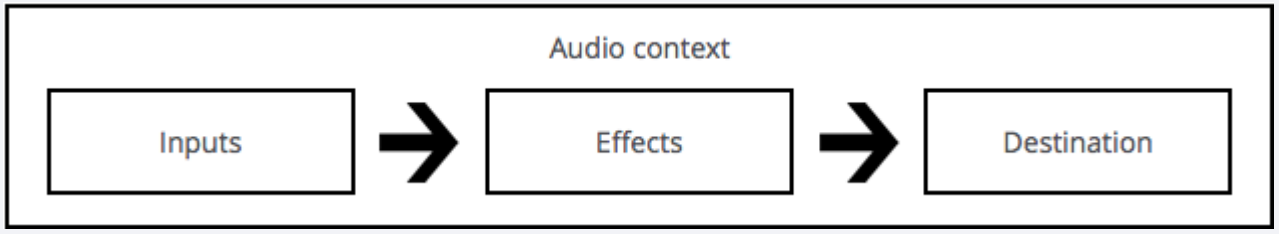
Chargement de fichiers audio puis création de contexte pour la lecture

Grande précision / faible latence dans le timing de lecture

Ajout d'effets (stéréo, echo, reverb, filtres, etc…)

02 GESTION DU SON

1. Création d'un contexte audio
2. Création de source
3. Création de nœuds (pour les effets)
4. Choix de la sortie audio (enceintes ou autre nœud)
5. Connexion des nœuds aux effets, puis des effets à la destination



02 GESTION DU SON

Gestionnaire audio personnalisé

```
(function (window) {
    'use strict';

    function AudioManager(soundDescriptors, mute) {
        this.sounds = {};
        this.soundDescriptors = soundDescriptors || {};
        this.mute = mute;

        try {
            window.AudioContext = window.AudioContext || window.webkitAudioContext;
            this.audioContext = new window.AudioContext();
        } catch (e) {
            window.alert('API Audio non supportée.');
        }
    }

    AudioManager.prototype.loadSounds = function (soundDescriptors) {

        var promises = soundDescriptors.map(function (soundDescriptor) {
            var sound = this.sounds[soundDescriptor.title];

            if (!sound) {
                this.sounds[soundDescriptor.title] = {
                    title: soundDescriptor.title,
                    url: soundDescriptor.url,
                    initialGain: soundDescriptor.initialGain || 1
                };
                return this.loadSound(this.sounds[soundDescriptor.title]);
            }
            return sound;
        }.bind(this));

        return Promise.all(promises);
    };
```

02 GESTION DU SON

Chargement d'un son

```
36  AudioManager.prototype.loadSound = function (sound) {
37
38      return new Promise(function (resolve, reject) {
39
40          if (sound.buffer) {
41              resolve(sound);
42          } else {
43              var request = new XMLHttpRequest();
44              request.open('GET', sound.url, true);
45              request.responseType = 'arraybuffer';
46
47              request.onload = function () {
48                  this.audioContext.decodeAudioData(
49                      request.response,
50                      function (buffer) {
51                          sound.buffer = buffer;
52                          resolve(sound);
53                      }.bind(this),
54                      function (error) {
55                          console.error('Fail to decodeAudioData [%s]', sound.url, error);
56                          reject(sound);
57                      }
58                  );
59              }.bind(this);
60
61              request.onerror = function (event) {
62                  reject(new Error("Erreur XMLHttpRequest", event));
63              };
64
65              request.send();
66          }
67
68      }.bind(this));
69  };
```

02 GESTION DU SON

Lecture d'un son

```
76      AudioManager.prototype.play = function (sound, loop, loopStart, loopEnd) {
77          return new Promise(function (resolve) {
78              var source = this.audioContext.createBufferSource();
79              sound.source = source;
80
81              source.buffer = sound.buffer;
82
83              // Controle du volume
84              var gainNode = this.audioContext.createGain ? this.audioContext.createGain() : this.audioContext.createG
85              source.connect(gainNode);
86              gainNode.connect(this.audioContext.destination);
87              sound.gainNode = gainNode;
88              gainNode.gain.value = this.mute ? -1 : sound.initialGain;
89
90              // Gestion des boucles sonores
91              source.loop = loop;
92              if (loop) {
93                  source.loopStart = loopStart;
94                  source.loopEnd = loopEnd;
95
96                  resolve(source);
97              }
98
99              source.onended = function () {
100                 if (!loop) {
101                     resolve(source);
102                 }
103             };
104
105             source.connect(this.audioContext.destination);
106             source.start(0);
107         }.bind(this));
108     };
```

02 GESTION DU SON

Contrôle du volume

```
110     AudioManager.prototype.muteSound = function (sound) {
111         if (sound.source) {
112             sound.gainNode.gain.value = -1;
113         }
114     };
115     AudioManager.prototype.resumeSound = function (sound) {
116         if (sound.source) {
117             sound.gainNode.gain.value = sound.initialGain;
118         }
119     };
120
121     AudioManager.prototype.stopSound = function (sound) {
122         if (sound.source) {
123             sound.source.stop(0);
124         }
125     };
126
```

02 GESTION DU SON

Lecture des sons

```
131     AudioManager.prototype.stageBgm = function () {
132         return this.play(this.sounds.stage, true, 0, 0);
133     };
134
135     AudioManager.prototype.laser = function () {
136         return this.play(this.sounds.laser);
137     };
138
139     AudioManager.prototype.boom = function () {
140         return this.play(this.sounds.boom);
141     };
142
143     AudioManager.prototype.foom = function () {
144         return this.play(this.sounds.foom);
145     };
146
147     AudioManager.prototype.toogleMute = function () {
148         this.mute = !this.mute;
149         if (this.mute) {
150             this.muteSound(this.sounds.stage);
151         } else {
152             this.resumeSound(this.sounds.stage);
153         }
154         return this.mute;
155     };
```

03 GESTION DES COMMANDES

03 GESTION DES COMMANDES

Gamepad API pour gérer une manette

Keyboard events pour le clavier

Détection d’appui / relâche d’une touche et mise à jour de l’état

03 GESTION DES COMMANDES

Gestionnaire personnalisé pour le clavier

```
(function (window, document) {
    'use strict';

    function Keyboard(mapping) {
        this.keys = mapping || this.defaultMapping;

        this.actions = {};

        this.currentOnKeyDownListener = null;
        this.currentOnKeyUpListener = null;
    }

    Keyboard.prototype.defaultMapping = {
        // Space
        SHOOT: 32,
        LEFT: 37,
        UP: 38,
        RIGHT: 39,
        DOWN: 40,
        // Enter
        START: 13,
    };

    Keyboard.prototype.startDetection = function () {
        this.detection(this.onKeyDownListenerGame.bind(this), this.onKeyUpListenerGame.bind(this));
    };

    Keyboard.prototype.stopDetection = function () {
        this.detection();
    };
```

03 GESTION DES COMMANDES

Détection des touches

```
33      Keyboard.prototype.detection = function (onKeyDownListener, onKeyUpListener) {
34
35          document.removeEventListener('keydown', this.currentOnKeyDownListener);
36          this.currentOnKeyDownListener = null;
37          if (onKeyDownListener) {
38              this.currentOnKeyDownListener = onKeyDownListener;
39              document.addEventListener('keydown', this.currentOnKeyDownListener);
40          }
41
42          document.removeEventListener('keyup', this.currentOnKeyUpListener);
43          this.currentOnKeyUpListener = null;
44          if (onKeyUpListener) {
45              this.currentOnKeyUpListener = onKeyUpListener;
46              document.addEventListener('keyup', this.currentOnKeyUpListener);
47          }
48      };
49
50
51      Keyboard.prototype.onKeyDownListenerGame = function (event) {
52          this.keyControl(event, true);
53      };
54
55      Keyboard.prototype.onKeyUpListenerGame = function (event) {
56          this.keyControl(event, false);
57      };
```

03 GESTION DES COMMANDES

Mise à jour de l'état des touches

```
59      Keyboard.prototype.keyControl = function (event, state) {
60
61          if (event.keyCode === this.keys.LEFT) {
62              this.actions.LEFT = state;
63          } else if (event.keyCode === this.keys.RIGHT) {
64              this.actions.RIGHT = state;
65          }
66
67          if (event.keyCode === this.keys.UP) {
68              this.actions.UP = state;
69          } else if (event.keyCode === this.keys.DOWN) {
70              this.actions.DOWN = state;
71          }
72
73          if (event.keyCode === this.keys.SHOOT) {
74              this.actions.SHOOT = state;
75          }
76
77          if (event.keyCode === this.keys.START) {
78              this.actions.START = state;
79          }
80      };
81
82      window.Keyboard = Keyboard;
83
84  })(window, document);
```

03 GESTION DES COMMANDES

Un peu de logique pour détecter si le vaisseau peut bouger

-> On ne peut pas aller à gauche si x < 0

-> On ne peut pas aller à droite si x > taille du canvas

-> On ne peut pas aller en bas si y > hauteur du canvas

-> On ne peut pas aller à en haut si y < 0

03 GESTION DES COMMANDES

Application des commandes

```
Game.prototype.checkInputInGame = function (control, physics, player) {

    // lance un shoot
    if (control.actions.SHOOT) {
        // création de 2 lasers
        this.lasersManager.shoot(player);

        // empêche le tir continue
        control.actions.SHOOT = false;
    }

    // déplacement horizontal
    if (control.actions.LEFT) {
        physics.moveLeft(player);
    } else if (control.actions.RIGHT) {
        physics.moveRight(player);
    }

    // déplacement vertical
    if (control.actions.UP) {
        physics.moveUp(player);
    } else if (control.actions.DOWN) {
        physics.moveDown(player);
    }
};
```

03 GESTION DES COMMANDES

Détection des déplacements

```
Physics.prototype.canMoveLeft = function (spaceship) {
    return spaceship.x >= 0 + this.moveSize + (spaceship.frameSize.width / 2);
};

Physics.prototype.moveLeft = function (spaceship) {
    if (this.canMoveLeft(spaceship)) {
        spaceship.x -= this.moveSize;
    }
};

Physics.prototype.canMoveRight = function (spaceship) {
    return spaceship.x <= this.canvasSize.width - this.moveSize - (spaceship.frameSize.width / 2);
};

Physics.prototype.moveRight = function (spaceship) {
    if (this.canMoveRight(spaceship)) {
        spaceship.x += this.moveSize;
    }
};
```

04 AUTOMATISATION DES ENNEMIS

04 AUTOMATISATION DES ENNEMIS

Qu’est-ce qu’une Promise ?

Effectuer une action asynchrone

Possibilité de chainer les actions

Evite l’enfer des callbacks

```
asyncCall(function(err, data1){
    if(err) return callback(err);
    anotherAsyncCall(function(err2, data2){
        if(err2) return calllback(err2);
        oneMoreAsyncCall(function(err3, data3){
            if(err3) return callback(err3);
            // are we done yet?
        });
    });
});
```

04 AUTOMATISATION DES ENNEMIS

Avec une Promise

```
asyncCall()
.then(function(data1){
    // do something...
    return anotherAsyncCall();
})
.then(function(data2){
    // do something...
    return oneMoreAsyncCall();
})
.then(function(data3){
   // the third and final async response
})
.fail(function(err) {
   // handle any error resulting from any of the above calls
})
.done();
```

04 AUTOMATISATION DES ENNEMIS

Création d'un scénario en JSON pour chaque ennemi

Chaque ennemi et son scénario est une Promise

On les met dans des groupes

On exécute les groupes uns par uns

04 AUTOMATISATION DES ENNEMIS

Partition des ennemis

```
{
    "groups" : [
        {
            "ships":
                [
                    [
                        {"id": 1, "action": "new", "x": 200, "y": -20, "type": "green", "life": 3, "speed": 3},
                        {"id": 1, "action": "move", "x": 200, "y": 300},
                        {"id": 1, "action": "shoot", "x": 200, "y": 640},
                        {"id": 1, "action": "move", "x": -10, "y": 300},
                        {"id": 1, "action": "leave"}
                    ],
                    [
                        {"id": 2, "action": "new", "x": 240, "y": -20, "type": "green", "life": 3, "speed": 3},
                        {"id": 2, "action": "move", "x": 240, "y": 300},
                        {"id": 2, "action": "shoot", "x": 240, "y": 640},
                        {"id": 2, "action": "move", "x": 500, "y": 300},
                        {"id": 2, "action": "leave"}
                    ]
                ]
        },
        {
            "ships":
                [
                    [
                        {"id": 7, "action": "new", "x": 0, "y": 80, "type": "red", "life": 3, "speed": 3},
                        {"id": 7, "action": "shoot", "x": 300, "y": 640},
                        {"id": 7, "action": "move", "x": 400, "y": 80},
                        {"id": 7, "action": "leave"}
                    ],
                    [
```

04 AUTOMATISATION DES ENNEMIS

Démarrage du niveau

```
1  EnnemiesManager.prototype.start = function (scenario) {
2
3      var groupSequence = Promise.resolve();
4
5      scenario.groups.every(function (group) {
6
7          var playing = this.gameState.isPlaying();
8          if (playing) {
9              groupSequence = groupSequence.then(function () {
10                 return this.startGroup(group);
11             }.bind(this));
12         }
13         // stop when game is not playing
14         return playing;
15
16     }.bind(this));
17
18     return groupSequence;
19 };
```

04 AUTOMATISATION DES ENNEMIS

Démarrage d'un groupe d'ennemis

```
1  EnnemiesManager.prototype.startGroup = function (group) {
2
3      var ships = [];
4
5      group.ships.every(function (ship) {
6
7          var playing = this.gameState.isPlaying();
8          if (playing) {
9              ships.push(this.commandShip(ship));
10         }
11         // stop when game is not playing
12         return playing;
13
14     }.bind(this));
15
16     return Promise.all(ships);
17 };
```

04 AUTOMATISATION DES ENNEMIS

Actions des ennemis

```
EnnemiesManager.prototype.commandShip = function (commands) {
        var sequence = Promise.resolve();

        commands.forEach(function (command) {
            sequence = sequence.then(function () {
                switch (command.action) {
                case 'new':
                    return this.commandNew(command);
                case 'move':
                    return this.commandMove(command);
                case 'shoot':
                    return this.commandShoot(command);
                case 'leave':
                    return this.commandLeave(command);
                default:
                    throw new Error('Invalid Ennemy ship command');
                }
            }.bind(this));

        }.bind(this));

        return sequence;
    };
```

04 AUTOMATISATION DES ENNEMIS

Apparition d'un ennemi

```
1  EnnemiesManager.prototype.commandNew = function (commands) {
2      var promise = Promise.resolve();
3      if (this.gameState.isPlaying()) {
4
5          var ennemy;
6
7          switch (commands.type) {
8          case 'red':
9              ennemy = new EnnemyRed(commands, this.resources);
10             break;
11         case 'green':
12             ennemy = new EnnemyGreen(commands, this.resources);
13             break;
14         case 'blue':
15             ennemy = new EnnemyBlue(commands, this.resources);
16             break;
17         default:
18             throw new Error('Invalid Ennemy ship type');
19         }
20
21         this.ennemies[commands.id] = ennemy;
22         promise = ennemy.startLoop(ennemy.FLY);
23     }
24     return promise;
25 };
```

04 AUTOMATISATION DES ENNEMIS

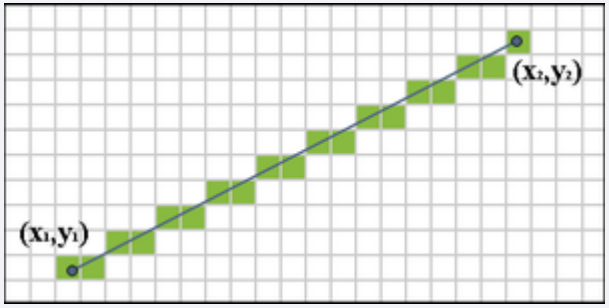
Déplacement d'un ennemi

```
1  EnnemiesManager.prototype.commandMove = function (commands) {
2      var promise = Promise.resolve();
3      var ennemy = this.ennemies[commands.id];
4
5      // Si le vaisseau n'a pas explosé
6      if (ennemy && this.gameState.isPlaying()) {
7          promise = this.pathManager.buildPath(ennemy, commands)
8              .then(function (path) {
9                  if (this.gameState.isPlaying()) {
10                     return ennemy.move(path);
11                 }
12                 return Promise.resolve();
13             }.bind(this));
14     }
15     return promise;
16 };
```

04 AUTOMATISATION DES ENNEMIS

Les ennemis bougent d'un point A vers un point B

Détermination du chemin avec l'algorithme de Bresenham



04 AUTOMATISATION DES ENNEMIS

Calcul de trajectoire

```
PathManager.prototype.buildPath = function (from, to) {
    return new Promise(function (resolve) {
        var coordinates = [];

        var dx = Math.abs(to.x - from.x),
            sx = from.x < to.x ? 1 : -1;
        var dy = Math.abs(to.y - from.y),
            sy = from.y < to.y ? 1 : -1;
        var err = (dx > dy ? dx : -dy) / 2;

        while (from.x !== to.x || from.y !== to.y) {
            coordinates.push({
                x: from.x,
                y: from.y
            });
            var e2 = err;
            if (e2 > -dx) {
                err -= dy;
                from.x += sx;
            }
            if (e2 < dy) {
                err += dx;
                from.y += sy;
            }
        }
        resolve(coordinates);
    });
};
```

05 HITBOXES ET COLLISIONS

05 HITBOXES ET COLLISIONS

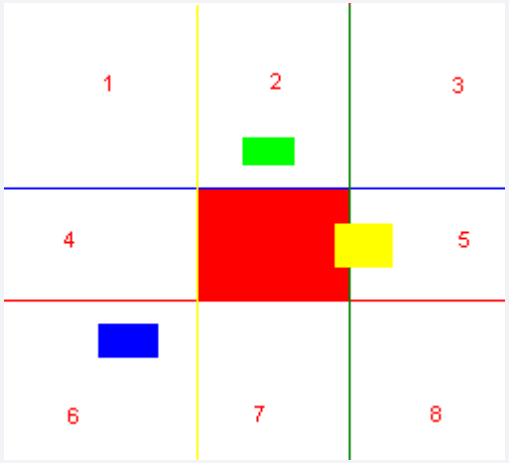
Hitbox : zone sensible à une collision

Détecter lorsque des balles touchent le joueur ou un vaisseau ennemi

AABB : Axis Aligned Bounding Box

Vérification régulière de collision

05 HITBOXES ET COLLISIONS



Le rectangle rouge est box1

Un rectangle box2 ne touche pas si :

- il est à gauche de la ligne jaune
- il est à droite de la ligne verte
- il est en haut de la ligne bleue
- il est en bas de la ligne rouge

Bleu ne touche pas
Vert ne touche pas
Jaune touche

05 HITBOXES ET COLLISIONS

Hitbox

```
/**
 * HitBox
 *
 *       maxY
 *        |
 * minX---|---maxX
 *        |
 *       minY
 *
 * @param {Number} minX Position X minimum
 * @param {Number} maxX Position X maximum
 * @param {Number} minY Position Y minimum
 * @param {Number} maxY Position Y maximum
 */
function HitBox(minX, maxX, minY, maxY) {
    this.minX = minX || 0;
    this.maxX = maxX || 0;
    this.minY = minY || 0;
    this.maxY = maxY || 0;
}

/**
 * Evaluation d'une collision avec une autre HitBox
 * @param   {Object}  hitbox HitBox à tester
 * @returns {Boolean} true si la HitBox est en collision avec la HitBox à tester
 */
HitBox.prototype.collision = function (hitbox) {
    return this.minX < hitbox.maxX &&
        this.maxX > hitbox.minX &&
        this.minY < hitbox.maxY &&
        this.maxY > hitbox.minY;
};
```

05 HITBOXES ET COLLISIONS

Si collision entre un laser du joueur et un ennemi
-> l'ennemi perd un point de vie jusqu'à ce qu'il explose

Si collision entre un laser ennemi et le joueur
-> on fera exploser le joueur, puis fin de partie

05 HITBOXES ET COLLISIONS

```
Physics.prototype.detectCollisionOnEnnemies = function (ennemies, playersLasers) {

    var ennemiesToDelete = [];
    var lasersToDelete = [];
    var score = 0;

    for (var id in ennemies) {
        if (ennemies.hasOwnProperty(id)) {
            var ennemy = ennemies[id];

            for (var j = 0; j < playersLasers.length; j++) {
                var laser = playersLasers[j];

                // collisions basées sur boites englobantes
                if (ennemy.hitbox().collision(laser.hitbox())) {

                    // Calcul du dommage
                    ennemy.life -= laser.strength;

                    // Ennemy mort
                    if (ennemy.life <= 0 && ennemiesToDelete.indexOf(ennemy) === -1) {
                        // Pour éviter de supprimer 2 fois le même (2 lasers peuvent être en même temps en collision sur un va
                        ennemiesToDelete.push(ennemy);
                    }

                    lasersToDelete.push(laser);
                }
            }

            for (var i = 0; i < lasersToDelete.length; i++) {
                var laserIndex = playersLasers.indexOf(lasersToDelete[i]);
                playersLasers.splice(laserIndex, 1);
            }
        }

    }

    this.ennemyDeath(ennemies, ennemiesToDelete);

    return score + 10 * ennemiesToDelete.length;
};
```

05 HITBOXES ET COLLISIONS

```
Physics.prototype.detectCollisionsOnPlayer = function (player, bullets, ennemies) {

    var collision = false;

    // collision avec les bullets
    var bulletsToDelete = [];
    bullets.forEach(function (bullet) {

        // collisions basées sur boites englobantes
        if (player.hitbox().collision(bullet.hitbox())) {
            collision = true;
            bulletsToDelete.push(bullet);
        }

    });

    for (var i = 0; i < bulletsToDelete.length; i++) {
        delete bullets[bulletsToDelete[i].id];
    }

    // collision avec les ennemies
    if (!collision) {

        var ennemiesToDelete = [];
        for (var id in ennemies) {
            if (ennemies.hasOwnProperty(id)) {
                var ennemy = ennemies[id];
                if (player.hitbox().collision(ennemy.hitbox())) {
                    collision = true;
                    ennemiesToDelete.push(ennemy);
                }
            }
        }

        this.ennemyDeath(ennemies, ennemiesToDelete);
    }

    return collision;
};
```

06 ORCHESTRATION DU JEU

06 ORCHESTRATION DU JEU

Chargement des ressources :

-> images

-> audio

-> scénario

Initialisation de toutes les fonctionnalités

Démarrage de la partie

06 ORCHESTRATION DU JEU

Launcher

```
window.requestAnimationFrame = (function () {
    // La fonction d'origine que tous les navigateurs finiront par utiliser.
    return window.requestAnimationFrame ||
        // Pour Chrome et Safari.
        window.webkitRequestAnimationFrame ||
        // Pour Firefox.
        window.mozRequestAnimationFrame ||
        // Pour Opera.
        window.ORequestAnimationFrame ||
        // Pour Internet Explorer.
        window.msRequestAnimationFrame ||
        function (callback) {
            window.setTimeout(callback, 1000 / 60);
        };
})();

var canvas = document.getElementById('game');
var context2d = canvas.getContext('2d');

var canvasSize = {
    // 640
    height: canvas.height,
    // 480
    width: canvas.width
};

var sprites = [
    {title: 'spaceship-red', url: 'resources/image/spaceship-red.png'},
    {title: 'spaceship-blue', url: 'resources/image/spaceship-blue.png'},
    {title: 'spaceship-green', url: 'resources/image/spaceship-green-large.png'},
    {title: 'ennemy-green', url: 'resources/image/ennemy-green.png'},
    {title: 'ennemy-red', url: 'resources/image/ennemy-red.png'},
```

06 ORCHESTRATION DU JEU

Launcher

```
37      {title: 'sky', url: 'resources/image/sky.jpg'},
38      {title: 'bullet', url: 'resources/image/bullet.png'},
39      {title: 'galaxy', url: 'resources/image/galaxy3.jpg'},
40      {title: 'stars', url: 'resources/image/stars2.png'}
41  ];
42  var resources = new Resources(sprites);
43
44  var mute = window.localStorage.getItem('mute');
45
46  var sounds = [
47      {title: 'stage', url: 'resources/audio/loop.mp3', initialGain: -0.7},
48      {title: 'laser', url: 'resources/audio/science_fiction_laser_005.mp3', initialGain: -0.8},
49      {title: 'boom', url: 'resources/audio/DeathFlash.ogg', initialGain: 1},
50      {title: 'foom', url: 'resources/audio/foom_0.ogg', initialGain: 1}
51  ];
52  var audioManager = new AudioManager(sounds, mute);
53
54  // Button on/off pour le son
55  var btnMute = document.getElementById('mute');
56  btnMute.onclick = function () {
57      audioManager.toogleMute();
58      this.classList.toggle('btn-mute-on');
59      this.classList.toggle('btn-mute-off');
60      window.localStorage.setItem('mute', !window.localStorage.getItem('mute'));
61  };
62  if (mute) {
63      btnMute.classList.toggle('btn-mute-on');
64      btnMute.classList.toggle('btn-mute-off');
65  }
66
```

06 ORCHESTRATION DU JEU

Launcher

```
67  var gameState = new GameState();
68
69  var controlsP1 = new Keyboard();
70
71  var pathManager = new PathManager();
72  var explosionManager = new ExplosionManager(resources, audioManager);
73
74  var bulletsManager = new BulletsManager(resources, pathManager);
75  var ennemiesManager = new EnnemiesManager(gameState, resources, pathManager, bulletsManager);
76
77  var physics = new Physics(canvasSize, explosionManager);
78  var lasersManager = new LasersManager(resources, canvasSize, audioManager);
79  var playerFactory = new PlayerFactory(resources, canvasSize);
80
81  var background;
82
83  var preLoadActions = [
84      // préchargement des sons
85      audioManager.load(),
86
87      // préchargement des images
88      resources.load()
89  ];
```

06 ORCHESTRATION DU JEU

Launcher

```
91  Promise.all(preLoadActions)
92      .then(function () {
93
94          // contruction du fond
95          background = new Background(resources, canvasSize);
96
97          // démarrage de la musique de fond
98          return audioManager.stageBgm();
99      })
100     .then(function () {
101
102         // construction du scénario des ennemies
103         return ennemiesManager.loadScenario('resources/stage1.json');
104     }).then(function (scenario) {
105
106         // démarrage de gestion des controles utilisateurs
107         controlsP1.startDetection();
108
109         // Création du jeux
110         var game = new Game(canvas, context2d, gameState, explosionManager, bulletsManager, ennemiesManager, lase
111
112         // affichage du jeux
113         game.paint();
114
115         // démarrage du jeux
116         game.start(scenario);
117     });
```

06 ORCHESTRATION DU JEU

Comment le jeu se termine ?

Plusieurs états : en chargement, en jeu, perdu, terminé, …

Dans la boucle d’affichage, détection de cet état

Si on arrive à la fin du scénario : état terminé

06 ORCHESTRATION DU JEU

Boucle principale

```
4   Game.prototype.paint = function () {
5
6       this.background.paint(this.context2d);
7
8       if (this.gameState.isPlaying()) {
9
10          this.checkInputInGame(this.controlsP1, this.physics, this.player1);
11          this.ennemiesManager.paint(this.context2d);
12          this.explosionManager.paint(this.context2d);
13          this.bulletsManager.paint(this.context2d);
14          this.player1.paint(this.context2d);
15          this.lasersManager.paint(this.context2d);
16          this.paintScores(this.context2d, this.gameState.scores.player1);
17          this.gameState.scores.player1 += this.physics.detectCollisionOnEnnemies(this.ennemiesManager.ennemies, this.lasersMan
18
19          if (this.physics.detectCollisionsOnPlayer(this.player1, this.bulletsManager.bullets, this.ennemiesManager.ennemies))
20              this.destroyPlayer(this.player1);
21          }
22
23          this.lasersManager.move();
24
25      } else if (this.gameState.isGameOver()) {
26          this.explosionManager.paint(this.context2d);
27
28      } else if (this.gameState.isFinished()) {
29          this.paintEndScreen(this.context2d, this.gameState.scores.player1);
30          this.checkInputMenu(this.controlsP1);
31      } else {
32          console.log('Loading...');
33      }
34
35      window.requestAnimationFrame(this.paint.bind(this));
36  };
```

THE END

Thank you for playing