DÉVELOPPEMENT DE JEU HTML5 AVEC PHASER

Ana ROXIN

TP

1e partie

Introduction

Qu'est-ce que Phaser ?

- Nous allons utiliser la plateforme de développement de jeux Phaser afin de créer un jeu multiplateformes
- Phaser est une plateforme gratuite et rapide, permettant de créer des jeux HTML5 pour les navigateurs mobiles et de bureau
  - Supporte à la fois Canvas et WebGL
  - Téléchargeable depuis www.phaser.io

- Récupérez le fichier Bunny_Defender.zip
  - Bibliothèques JS Phaser
  - Images – extraites depuis animation Flash
  - Fichiers audio

Ouverture du projet

- Avec l'éditeur de votre choix (Eclipse, Aptana, UltraEdit, Sublime Text, etc.)
  - Créez un nouveau projet
- Dans le dossier du projet
  - Copiez l'ensemble des dossiers et fichiers de l'archive
  - BunnyDefender.zip (sur le serveur)
- Ouvrez ensuite index.html
  - Remarquez l'inclusion de la librairie complète phaser.js
    - On préfère la version complète plutôt que la version minimale car en cas de problème celle-ci est plus facile à débugguer

index.html

index.html

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="utf-8">
    <title>Bunny Defender</title>
    <script src="libs/phaser.js" type="text/javascript"></script>
    <style>
        body {
            padding: 0;
            margin: 0;
            background-color: #000000;
        }
    </style>
</head>
<body>
    <div id="gameContainer">
        <script type="text/javascript">
            window.onload = function() {
                //nouvelle variable = nouvelle instance de la classe Phaser.game
                //attributs = largeur, hauteur, moteur graphique (Canvas/WebGL), refr vers la div ou sera affiche le jeu
                var game = new Phaser.Game(540, 960, Phaser.AUTO, 'gameContainer');
            };

        </script>
    </div>

</body>
</html>
```

2e partie

Configuration des chargements et des états

Phase de démarrage - bootloader

- Le premier état du jeu c'est le chargement
  - Le bootloader de Phaser effectue des configurations supplémentaires de Phaser et prépare les élément pré-chargement (pre-loader assets)
- Nous allons créer le fichier JavaScript qui représente cet état
  - Nouveau fichier – boot.js
- On commence par déclarer une référence vers le jeu, que nous pourrons passer d'un état à l'autre
  - variable BunnyDefender = objet vide
- On déclare notre état sur BunnyDefender
  - BunnyDefender.boot = fonction qui prend en paramètre l'objet jeu (même objet que celui créé dans index.html)
- On fait le lien entre le fichier *.js et le fichier index.html

Boot.js

index.html | Boot.js

```
1  var BunnyDefender = {};
2
3  //l'objet Boot ici doit être identique au nom de fichier JS'
4  BunnyDefender.Boot = function() {
5      // fonction Phaser - essaie de pré-charger des éléments avant de les utiliser
6      preload: function() {},
7      // fonction create Phaser
8      create: function() {
9          //on définit l'input'
10         //this fait référence à l'objet jeu passé en paramètre
11         //on aura toujours qu'un seul pointeur activé'
12         this.input.maxPointers = 1;
13         //si on est dans un navigateur et on ouvre un autre onglet, cela met le jeu en pause
14         this.stage.disableVisibilityChange = false;
15         //affiche tout ce qui est défini dans la scène
16         this.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
17         //largeur minimum
18         this.scale.minWidth = 270;
19         //hauteur minimum
20         this.scale.minHeight = 480;
21         //centrage du jeu
22         this.scale.pageAlignHorizontally = true;
23
24         this.scale.pageAlignVertically = true;
25         //on force l'affichage en mode portrait'
26         this.stage.forcePortrait = true;
27         //on force la taille de l'écran de notre jeu'
28         this.scale.setScreenSize(true);
29
30         this.input.addPointer();
31         //on définit l'arrière plan sur du bleu'
32         this.stage.backgroundColor = '#171642';
33     }
34 }
```

index.html – v2

index.html × Boot.js

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="utf-8">
    <title>Bunny Defender</title>
    <script src="libs/phaser.js"></script>
    <script src="Boot.js"></script>
    <style>
        body {
            padding: 0;
            margin: 0;
            background-color: #000000;
        }
    </style>
</head>
<body>
    <div id="gameContainer">
        <script type="text/javascript">
            window.onload = function() {
                //nouvelle variable = nouvelle instance de la classe Phaser.game
                //attributs = largeur, hauteur, moteur graphique (Canvas/WebGL), refr vers la div ou sera affiche le jeu
                var game = new Phaser.Game(540, 960, Phaser.AUTO, 'gameContainer');
                //on précise l'état de notre jeu
                game.state.add('Boot', BunnyDefender.Boot);
            };

        </script>
    </div>

</body>
</html>
```

Phase de chargement - preloader

- Notre jeu dépend de plusieurs ressources externes (images, sons, etc.)
- La phase de (pré-)chargement veille à ce que l'ensemble des ressources est chargé et prêt à être utilisé avant de lancer d'autres états supplémentaires du jeu
- Dans Boot.js, fonction preload
  - Faire le chargement des ressources nécessaires au pré-chargement lui-même
  - Barre de progression, une image de titre
- Dans Boot.js, fonction create
  - On doit exécuter le (pré-)chargement – on doit créer un fichier Preloader.js
- Preloader.js
  - On initialise les variables pour la barre de progression, l'image de titre
  - On spécifie l'ensemble des fonctions Phaser pour le pré-chargement
- On doit encore faire le lien vers Preloader.js depuis index.html

Preloader.js

index.html | Boot.js | Preloader.js

```
1  //on passe toujours en paramètre notre objet jeu
2  BunnyDefender.Preloader = function(game) {
3      //on a besoin d'une barre de chargement + une image de titre
4      this.preloadBar = null;
5      this.titleText = null;
6      //on a rien chargé pour le moment, le jeu est pas prêt
7      this.ready = false;
8  };
9
10 BunnyDefender.Preloader.prototype = {
11
12     preload: function () {
13         //references vers la barre de progression en tant que sprite, puis la redimmensionne selon le centre du jeu vertical et horizontal
14         this.preloadBar = this.add.sprite(this.world.centerX, this.world.centerY, 'preloaderBar');
15         this.preloadBar.anchor.setTo(0.5, 0.5);
16         //associe le srpite créé au mécanisme de pré-chargement
17         this.load.setPreloadSprite(this.preloadBar);
18         //ref vers le texte, avec petit ajustement pour la position verticale
19         this.titleText = this.add.image(this.world.centerX, this.world.centerY-220, 'titleimage');
20         //on définit le point d'ancrage sur le centre de l'objet
21         this.titleText.anchor.setTo(0.5, 0.5);
22     },
23
24     create: function () {
25         //la barre de chargement est à 100% donc plus besoin de la réduire
26         this.preloadBar.cropEnabled = false; //force show the whole thing
27     },
28
29     update: function () {
30         //dès que tout est prêt, le jeu peut s'exécuter
31         this.ready = true;
32     }
33 };
```

index.html – v3

index.html × | Boot.js | Preloader.js

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="utf-8">
    <title>Bunny Defender</title>
    <script src="libs/phaser.js"></script>
    <script src="Boot.js"></script>
    <script src="Preloader.js"></script>
    <style>
        body {
            padding: 0;
            margin: 0;
            background-color: #000000;
        }
    </style>
</head>
<body>
    <div id="gameContainer">
        <script type="text/javascript">
            window.onload = function() {
                //nouvelle variable = nouvelle instance de la classe Phaser.game
                //attributs = largeur, hauteur, moteur graphique (Canvas/WebGL), refr vers la div ou sera affiche le jeu
                var game = new Phaser.Game(540, 960, Phaser.AUTO, 'gameContainer');
                //on précise l'état de notre jeu
                game.state.add('Boot', BunnyDefender.Boot);
                //permet d'avoir la phase de pré-chargement à l'intérieur du jeu
                game.state.add('Preloader', BunnyDefender.Preloader);
                //on veut démarer le jeu à partir de Boot.js
                game.state.start('Boot');
            };

        </script>
    </div>

</body>
</html>
```

Exécution

http://127.0.0.1:8020/Bunny_Defender/index.html
Mobile portrait (320x480)
Bunny
Defender

Construction du menu (1)

- Actuellement, notre jeu comprend 2 états:
  - Un qui sert simplement de chargement au démarrage
  - Un qui pré-charge nos éléments et informe l'utilisateur lorsque le jeu se charge

- Nous allons construire l'écran de démarrage pour notre jeu – il faut informer le joueur que tout est prêt
  - Avant cela, il faut terminer de charger quelques éléments supplémentaires
    - Une image + une polices de caractères

Construction du menu (2)

- Dans Preloader.js
  - Dans la fonction preload()
    - Ajouter le chargement de l'image images/TitleBG.png
      - `this.load.image('demarrage', 'images/TitleBG.png');)`
    - Ajouter le chargement de la police eightbitwonder
      - `this.load.bitmapFont('eightbitwonder', 'fonts/eighbitwonder.png', 'fonts/eightbitwonder.fnt');`
  - Dans la fonction update()
    - Ajouter le démarrage du menu, lorsque tout a été chargé
      - `this.state.start('StartMenu');`

- Créez un nouveau fichier JS, appellé StartMenu.js

StartMenu.js (1)

- Dans ce fichier, nous déclarons la fonction StartMenu de notre objet BunnyDefender
- Cette fonction contiendra 2 déclarations:
  - Un arrière-plan
    - `this.startBG;`
  - Une invite
    - `this.startPrompt;`

- Pour les fonctions de base de Phaser, il faut créer le prototype StartMenu, avec les fonctions spécifiques:
  - create (intégré à Phaser)
  - startGame (fonction personnalisée)

StartMenu.js (2)

index.html | Preloader.js | StartMenu.js

```
1  BunnyDefender.StartMenu = function(game) {
2      //déclaration de l'arrière-plan
3      this.startBG;
4      //invite
5      this.startPrompt;
6  };
7
8  BunnyDefender.StartMenu.prototype = {
9
10     create : function() {
11         startBG = this.add.image(0, 0, 'demarrage');
12         startBG.inputEnabled = true;
13         startBG.events.onInputDown.addOnce(this.startGame, this);
14
15         startPrompt = this.add.bitmapText(this.world.centerX - 155, this.world.centerY + 180, 'eightbitwonder', 'Appuyez pour démarrer!', 24);
16     },
17
18     startGame : function(pointer) {
19         this.state.start('Game');
20     }
21 };
22
```

Construction du menu (3)

- Il reste à attacher « StartMenu » à notre objet jeu, puis la charger en tant que phase de ce jeu
- Dans index.html
  - Ajouter la référence vers StartMenu.js
  - Dans le <body>, dans <script>
    - Ajouter le nouvel état ainsi créé
- Testez !

Exécution

Phase de jeu

- Nous sommes arrivés au point où nous devons concevoir la partie « jouable » du projet – ou la phase « Game »
  - Cet état comprendra l'ensemble des objets du jeu ainsi que sa logique
- A partir de StartMenu.js, on démarre l'état « Game » lorsque l'utilisateur clique sur l'écran de démarrage
- Créez le fichier Game.js
- Dans Game.js
  - Faites le lien avec l'objet BunnyDefender et l'objet « game » de index.html
  - Créez le prototype BunnyDefender.Game sur un objet avec 2 fonctions:
    - La fonction create de Phaser
    - La fonction update de Phaser
- Dans index.html, faites le lien avec le fichier Game.js

Game.js


```
BunnyDefender.Game = function(game) {};

BunnyDefender.Game.prototype = {
    create: function() {},
    update: function() {}
};
```

Ce sont les débuts de notre phase de jeu.
Il est possible d'ajouter d'autres états comme par exemple « GameOver » ou « MeilleursScores »

3e partie

Peupler le monde…

Preloader.js

- Maintenant que nous avons créé les états de notre jeu, nous pouvons peupler le jeu avec nos éléments de jeu
  - Pour ce faire, nous allons créer une fonction qui va créer ces éléments dans un ordre donné, en commençant par notre image d'arrière-plan
- Dans Preloader.js, dans la fonction preload – nous avons besoin de pré-charger les images pour le ciel et la colline, afin de les utiliser dans l'état « Game »
  - `this.load.image('colline', 'images/hill.png');`
  - `this.load.image('ciel', 'images/sky.png');`

Game.js (1)

- Dans la fonction create, créez une fonction buildWorld()
  - Cette fonction permettra d'organiser ce qui se passe dans l'état « Game »
- Nous allons commencer par ajouter les images du jeu pré-chargées dans Preloader.js
  - Une image pour le ciel (positionnée à 0, 0)
  - Une image pour la colline (positionnée à 0, 800)

Game.js (2)


```
BunnyDefender.Game = function(game) {};

BunnyDefender.Game.prototype = {
create: function() {

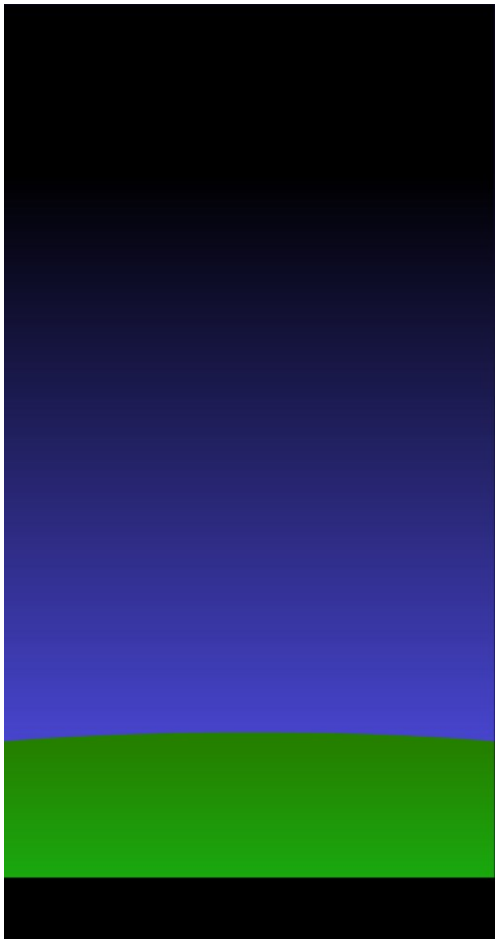
this.buildWorld();

},

buildWorld: function() {
//chargement image ciel
this.add.image(0,0,'ciel');
//chargement image colline
this.add.image(0,800,'colline');
},

update: function() {}
};
```

Exécution



Peuplement du monde créé (1)

- Pour générer et gérer les différentes créatures, nous allons utiliser les mécanismes de groupe et de phase
  - Lorsque des entités sont créées en tant que membres d'un groupe, il est possible d'y accéder individuellement mais aussi à un niveau de groupe

- Dans Preloader.js, dans preload
  - this.load.atlasXML('lapin', 'images/spritesheets/bunny.png', 'images/spritesheets/bunny.xml');
- Dans Game.js, ajoutez 2 attributs
  - this.nbLapins;
  - this.groupeLapins;

Peuplement du monde créé (2)

- Dans Game.js, dans la fonction

index.html | Preloader.js | Boot.js | StartMenu.js | Game.js

```
BunnyDefender.Game = function(game) {
    //nombre total de lapins
    this.nbLapins;
    //on controle des groupes de lapins
    this.groupeLapins;
};

BunnyDefender.Game.prototype = {
    create : function() {
        this.nbLapins = 20;
        this.buildWorld();

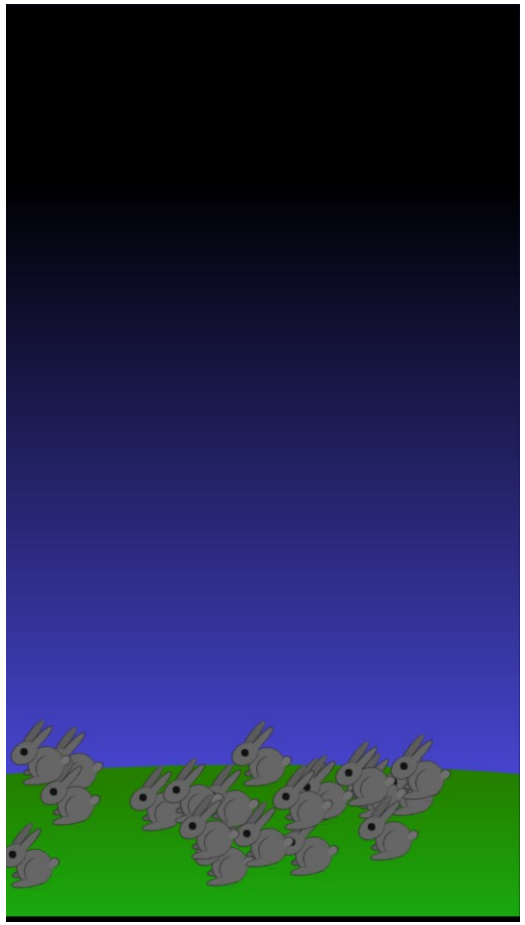
    },

    buildWorld : function() {
        //chargement image ciel
        this.add.image(0, 0, 'ciel');
        //chargement image colline
        this.add.image(0, 800, 'colline');
        //appel de la fonction qui va générer les lapins
        this.construireLapins();
    },

    construireLapins : function() {
        this.groupeLapins = this.add.group();
        this.groupeLapins.enableBody = true;
        for (var i = 0; i < this.nbLapins; i++) {
            var b = this.groupeLapins.create(this.rnd.integerInRange(-10, this.world.width - 50), this.rnd.integerInRange(this.world.height - 180, this.world.height - 60), 'lapin', 'Bunny0000');
            b.anchor.setTo(0.5, 0.5);
            b.body.moves = false;
            b.animations.add('Rest', this.game.math.numberArray(1, 58));
            b.animations.add('Walk', this.game.math.numberArray(68, 107));
            b.animations.play('Rest', 24, true);
        }
    },

    update : function() {
    }
};
```

Test



Déplacement des lapins

- Les lapins ainsi ajoutés ne font rien pour le moment
  - Le système d'interpolation de mouvement de Phaser couplé avec les 2 états d'animation définis précédemment peut résoudre ce problème
    - Il faut déclencher ces animations lorsque l'utilisateur interagit avec un lapin

- Plusieurs fonctions permettent de faire cela:
  - La première permet d'associer un mouvement aux lapins
  - La deuxième démarre le mouvement d'un lapin
  - La troisième arrête le mouvement d'un lapin

Associer un mouvement


```
38      associeMouvementLapin : function(b) {
39          bposition = Math.floor(this.rnd.realInRange(50, this.world.width - 50));
40          bdelay = this.rnd.integerInRange(2000, 6000);
41          if (bposition < b.x) {
42              b.scale.x = 1;
43          } else {
44              b.scale.x = -1;
45          }
46          t = this.add.tween(b).to({
47              x : bposition
48          }, 3500, Phaser.Easing.Quadratic.InOut, true, bdelay);
49          //properties, duration, ease, autoStart, delay
50          t.onStart.add(this.startBunny, this);
51          t.onComplete.add(this.stopBunny, this);
52      },
```

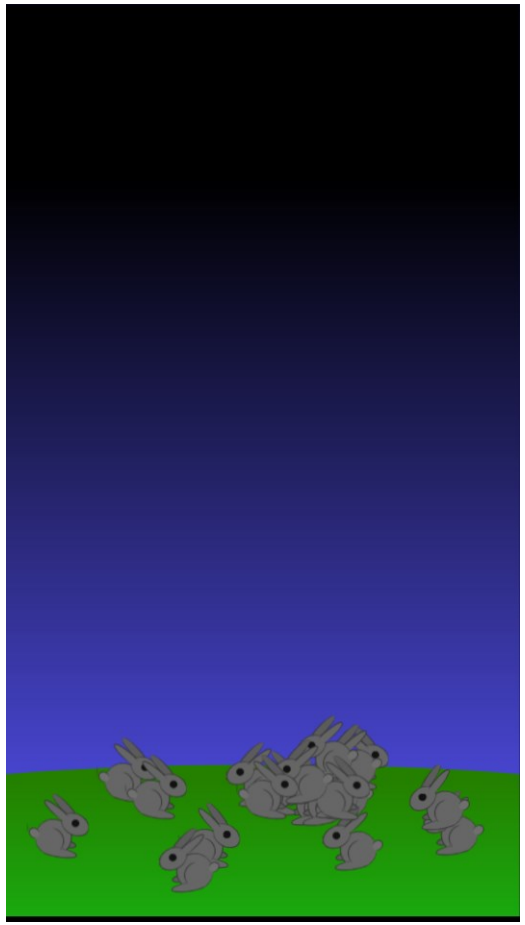
Appelez cette fonction à partir de la boucle for de la fonction construireLapins()

Démarrer / arrêter un mouvement


```
54      demarreLapin : function(b) {
55          b.animations.stop('Play');
56          b.animations.play('Walk', 24, true);
57      },
```

```
59      arreteLapin : function(b) {
60          b.animations.stop('Walk');
61          b.animations.play('Rest', 24, true);
62          this.associeMouvementLapin(b);
63      },
```

Test



Ajout d'ennemis (1)

- Notre jeu contient aussi des obstacles ou des ennemis contre lesquels il va falloir défendre les lapins
  - Il s'agira de pierres qui vont tomber du ciel, leur vitesse et taille étant variables
  - On utilisera le même mécanisme de groupement de Phaser que pour les lapins

- Chargement des pierres
  - Dans Preloader.js, après le chargement des lapins
  - ```
    this.load.atlasXML('pierre',
    'images/spritesheets/SpaceRock.png',
    'images/spritesheets/SpaceRock.xml');
    ```
    ```

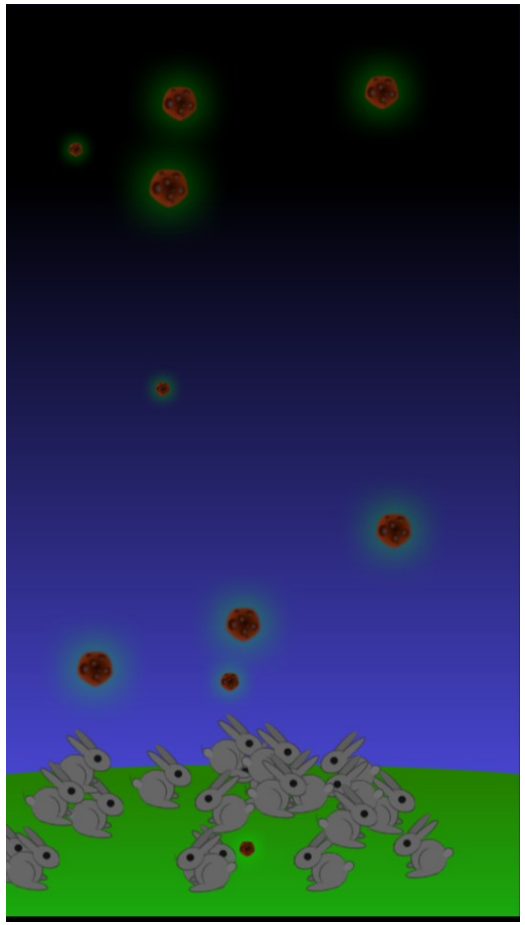
Ajout d'ennemis (2)

- Dans Game.js, déclaration de nouvelles variables
  - Fonction create: totalPierres et groupePierres
- Dans la fonction buildWorld() invoquer une fonction pour construire les pierres (après la construction des lapins)
  - Ajoutez le corps de cette fonction juste avant la fonction update

Ajout d'ennemis (3)


```
44      construirePierres : function() {
45          this.groupePierres = this.add.group();
46          for (var i = 0; i < this.totalPierres; i++) {
47              var p = this.groupePierres.create(this.rnd.integerInRange(0, this.world.width), this.rnd.realInRange(-1500, 0), 'pierre', 'SpaceRock0000');
48              var scale = this.rnd.realInRange(0.3, 1.0);
49              p.scale.x = scale;
50              p.scale.y = scale;
51              this.physics.enable(p, Phaser.Physics.ARCADE);
52              p.enableBody = true;
53              p.body.velocity.y = this.rnd.integerInRange(200, 400);
54              p.animations.add('Fall');
55              p.animations.play('Fall', 24, true);
56          }
57      },
```

Test



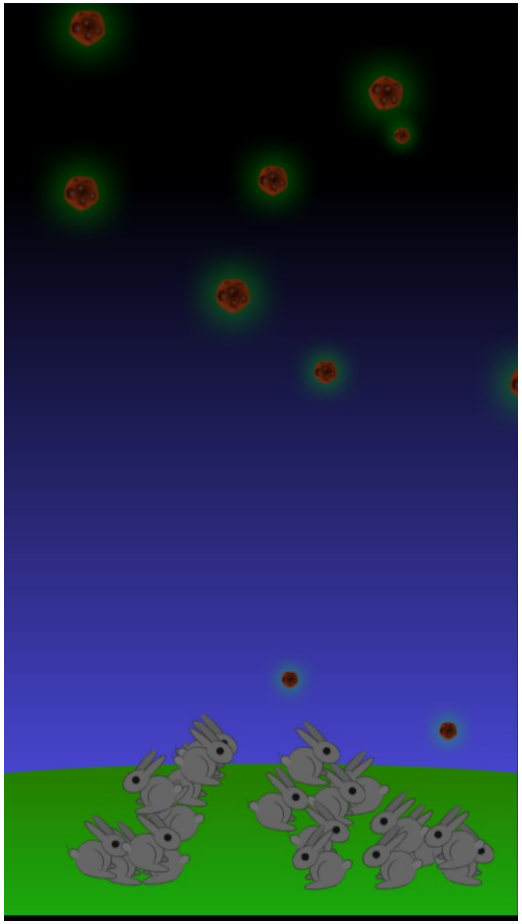
Vérifier les limites du monde

- Lorsqu'une pierre sort de l'écran, l'objet correspondant est perdu à jamais
  - Nous souhaitons le réutiliser, notamment en utilisant des mécanismes de Phaser
- Dans Game.js, dans la fonction construirePierres(), ajoutez les éléments suivants:
  - Une instruction disant à chacune des pierres de vérifier si elle a dépassé les limites du monde
    - On le définit à TRUE, donc lorsqu'une pierre sort du monde, elle lance un événement
  - Ensuite, on lie cet événement à la fonction this.resetPierre
    - This référence ici l'objet lui-même (à savoir la pierre)
  - Dans cette fonction, on ajoute une clause if pour tester si la position Y de la pierre considérée dépasse la hauteur du monde (les pierres ont donc traversé tout l'écran)
    - On appelle ici une nouvelle fonction this.recreePierre (ainsi on peut la réutiliser)

Game.js


```
44      construirePierres : function() {
45          //nouveau groupe
46          this.groupePierres = this.add.group();
47          //boucle sur l'ensemble des pierres (13 au total)
48          for (var i = 0; i < this.totalPierres; i++) {
49              //sur toute la largeur de la scène (position X des pierres), puis pour la position verticale à partir de -1500px (très au-dessus de la scène)
50              var p = this.groupePierres.create(this.rnd.integerInRange(0, this.world.width), this.rnd.realInRange(-1500, 0), 'pierre', 'SpaceRock0000');
51              //définition d'une échelle aléatoire
52              var scale = this.rnd.realInRange(0.3, 1.0);
53              //mise à l'échelle sur la hauteur et la largeur de la pierre
54              p.scale.x = scale;
55              p.scale.y = scale;
56              //on veut que les pierres tombent vers les lapins - on utilise la gravité de Phaser
57              this.physics.enable(p, Phaser.Physics.ARCADE);
58              p.enableBody = true;
59              //on définit une vitesse avec laquelle les pierres vont tomber
60              p.body.velocity.y = this.rnd.integerInRange(200, 400);
61              //les pierres n'ont qu'une seule animation - elles tombent / on ajoute l'animation, puis on la lit à 24 fps
62              p.animations.add('Fall');
63              p.animations.play('Fall', 24, true);
64              //vérification des limites du monde
65              p.checkWorldBounds = true;
66              p.events.onOutOfBounds.add(this.resetPierre, this);
67
68          }
69      },
70
71      resetPierre : function(p) {
72          if (p.y > this.world.height) {
73              this.recreePierre(p);
74          }
75      },
76
77      recreePierre : function(p) {
78          //on fait un reset Phaser - nouvelle position x et y
79          p.reset(this.rnd.integerInRange(0, this.world.width), this.rnd.realInRange(-1500, 0));
80          //idem pour la vitesse
81          p.body.velocity.y = this.rnd.integerInRange(200, 400);
82      },
```

Test



4e partie

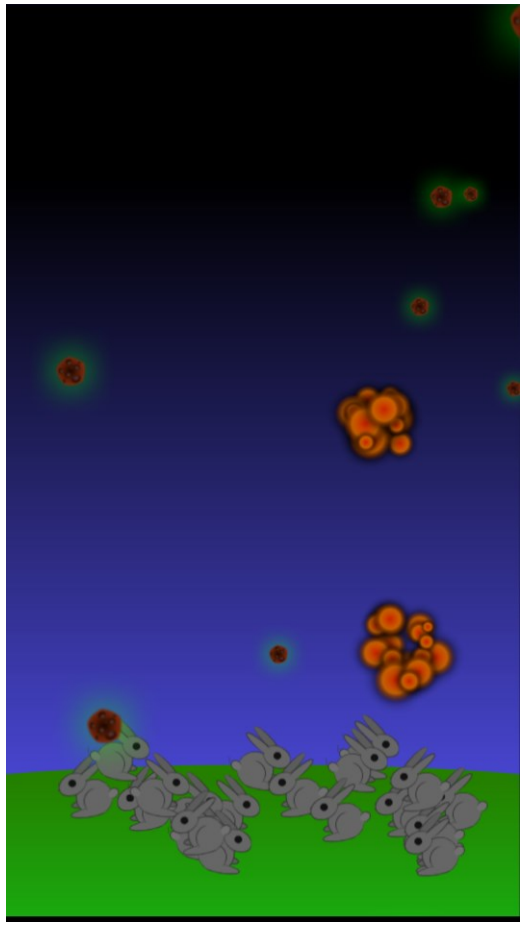
Fonctionnalités

Particules et émetteurs

- Dans Preloader.js :::
  - Fonction preload: this.load.image('explosion', 'images/explosion.png');
- Dans Game.js
  - this.eclat;
  - Fonction buildWorld: this.construireEmetteur();

```
86      construireEmetteur : function() {
87          this.eclat = this.add.emitter(0, 0, 80);
88          this.eclat.minParticleScale = 0.3;
89          this.eclat.maxParticleScale = 1.2;
90          this.eclat.minParticleSpeed.setTo(-30, 30);
91          this.eclat.maxParticleSpeed.setTo(30, -30);
92          this.eclat.makeParticles('explosion');
93          this.input.onDown.add(this.eclatFeu, this);
94      },
95
96      eclatFeu: function(pointer) {
97          this.eclat.emitX = pointer.x;
98          this.eclat.emitY = pointer.y;
99          this.eclat.start(true, 2000, null, 20);
100         //(explode, lifespan, frequency, quantity)
101     },
```

Test

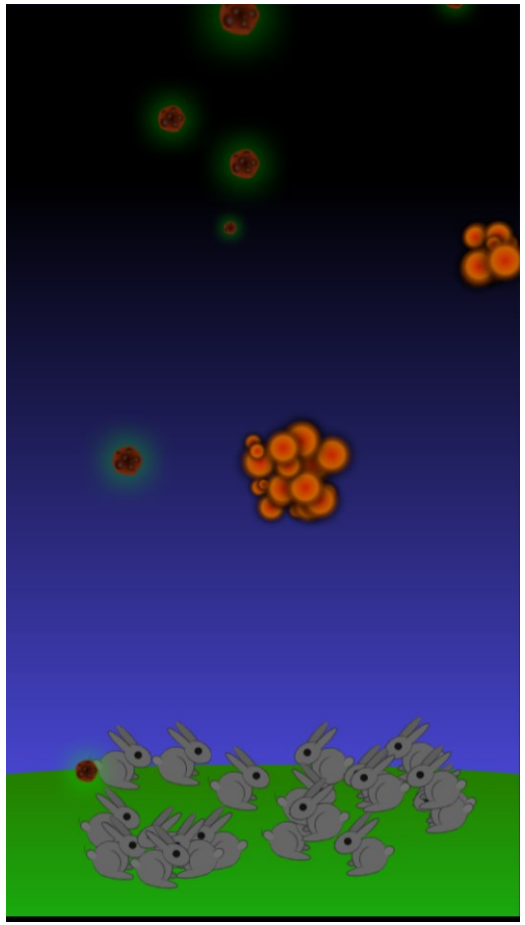


Gestion des collisions – souris / pierre

- Avec Phaser, la détection de collissions est rendue très facile, grâce à sa méthode update

- Dans Game.js
  - Fonction update
    - this.physics.arcade.overlap(this.groupePierres, this.eclat, this.eclatCollision, null, this);
  - Ajouter la fonction eclatCollision:
    - burstCollision: function(p, b) {
    - this.recreePierre(p);
    - },

Test

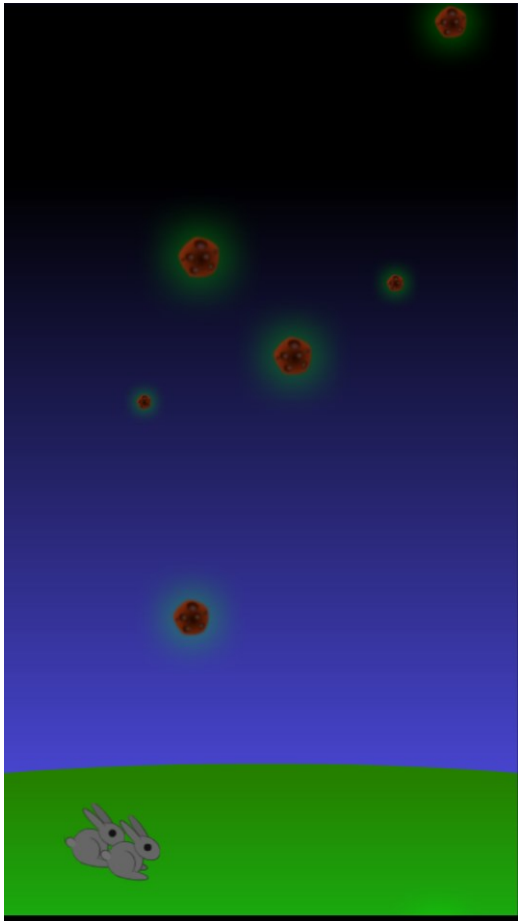


Gestion des collisions – lapin/pierre

- On doit réagir aux collisions entre les lapins et les pierres
  - Nous allons à nouveau utiliser la méthode update de Phaser
- Dans Game.js
  - Fonction update
    - this.physics.arcade.overlap(this.groupePierres, this.groupeLapins, this.lapinCollision, null, this);
  - Ajouter les fonctions lapinCollision et verifieLapinsRestants

```
lapinCollision: function(p, b) {
    if(b.exists){
        this.recreePierre(p);
        b.kill(); //méthode Phaser pour tuer un sprite
        this.nbLapins--;
        this.verifieLapinsRestants();
    }
},
verifieLapinsRestants: function() {
    if(this.nbLapins <= 0){
        // GAME OVER
    }
},
```

Test

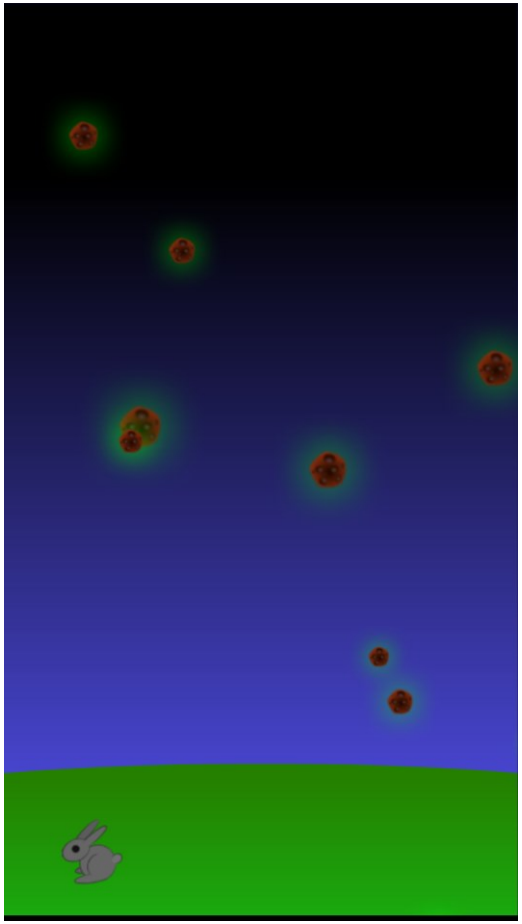


Tir ami

- On veut faire en sorte que les explosions puissent affecter les lapins
- Dans Game.js
  - Fonction update
    - this.physics.arcade.overlap(this.groupeLapins, this.eclat, this.tirAmi, null, this);
  - Ajouter la fonctions tirAmi:

```
tirAmi : function(b,e) {
        if (b.exists) {
            b.kill();
            this.nbLapins--;
            this.verifieLapinsRestants();
        }
},
```

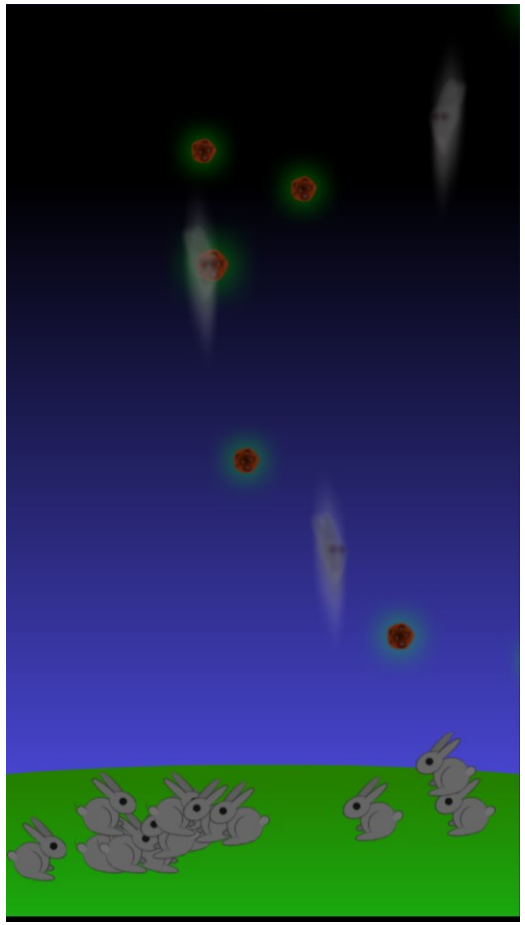
Test



Lapins fantômes

- Dans Preloader.js :::
  - Fonction preload
    - this.load.image('ghost', 'images/ghost.png');
- Dans Game.js
  - faireFantome: function(b) {
  - lapinfantome = this.add.sprite(b.x-20, b.y-180, 'ghost');
  - lapinfantome.anchor.setTo(0.5, 0.5);
  - lapinfantome.scale.x = b.scale.x;
  - this.physics.enable(lapinfantome, Phaser.Physics.ARCADE);
  - lapinfantome.enableBody = true;
  - lapinfantome.checkWorldBounds = true;
  - lapinfantome.body.velocity.y = -800;
  - },
  - -- lapinCollision
  - this.faireFantome(b); //avant de les tuer
  - -- tirAmi
  - this.makeGhost(b);

Test



5e partie

Avant de finir…

Détecter le « game over »

- Lorsque tous les lapins sont perdus dans des collisions…

- Dans Game.js
  - this.gameover;

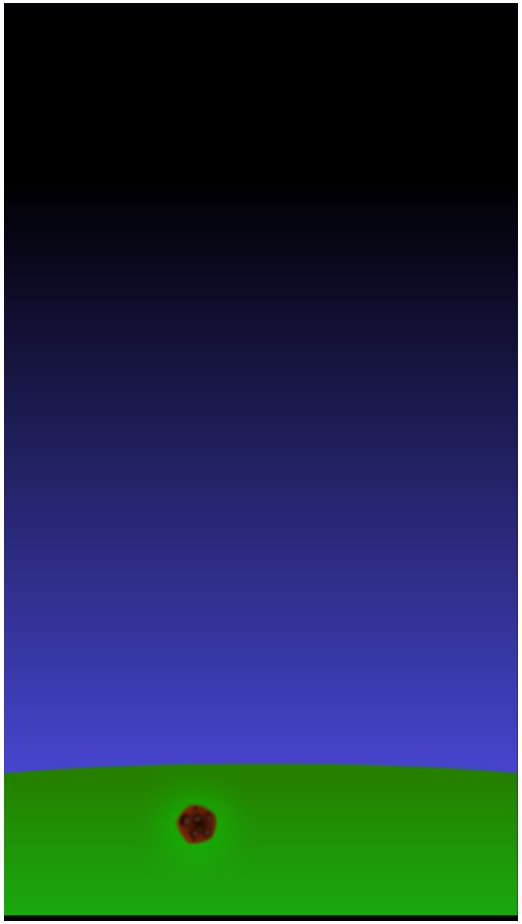
  - Dans la fonction create
    - this.gameover = false; (avant toute autre instruction)

  - Dans recreePierre
    - if(this.gameover == false){ /*code actuel*/}

  - Dans eclatFeu
    - if(this.gameover == false){ :µcode actuel*/}

  - Dans verifieLapinsRestants (à la place du commentaire)
    - this.gameover = true;

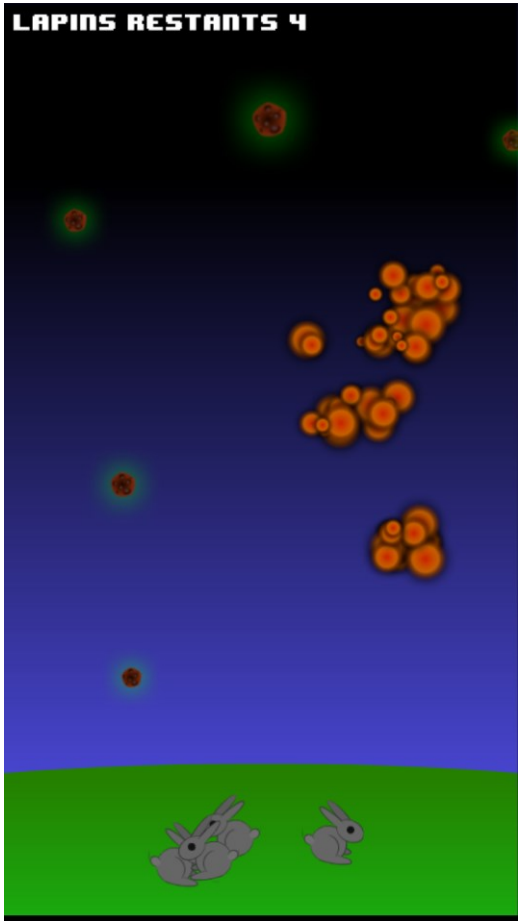
Test



Affichage du nombre de lapins restans

- Dans Game.js
  - this.countdown;
  - fonction buildWorld
    - this.countdown = this.add.bitmapText(10, 10, 'eightbitwonder', 'Bunnies Left ' + this.totalBunnies, 20);
  - Fonction verifieLapinsRestans
    - [if lapins == 0]
    - this.countdown.setText('Lapins restants 0');
    - [else]
    - this.countdown.setText('Lapins restants: ' + this.nbLapins);

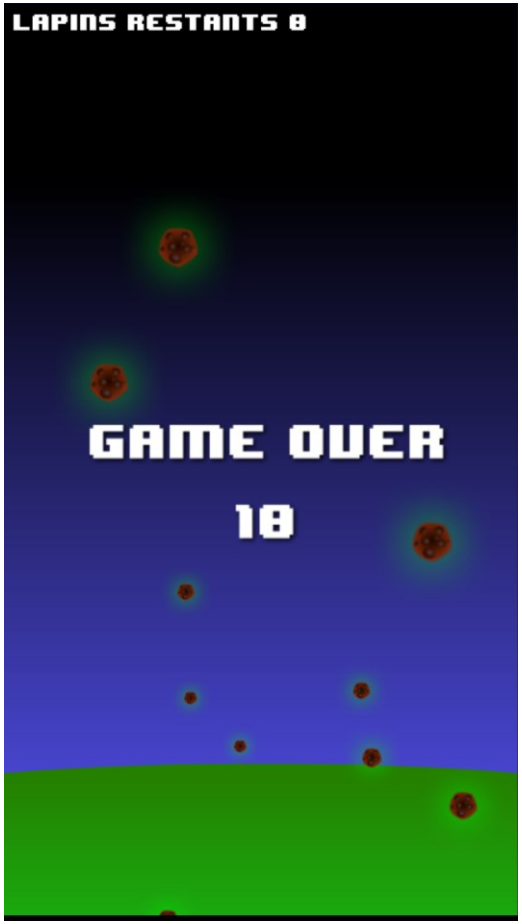
Test



Nombre de secondes jouées

- Dans Game.js
  - this.overmessage;
  - this.secondsElapsed;
  - this.timer;
  - Fonction create
    - this.secondsElapsed = 0;
    - this.timer = this.time.create(false);
    - this.timer.loop(1000, this.updateSeconds, this);
    - ADD TO Game.js
    - updateSeconds: function() {
    - this.secondsElapsed++;
    - },
  - Fonction buildWorld
    - this.timer.start();
    - -- checkBunniesLeft[if bunnies is 0]
    - this.overmessage = this.add.bitmapText(this.world.centerX-180, this.world.centerY-40, 'eightbitwonder', 'GAME OVER\n\n' + this.secondsElapsed, 42);
    - this.overmessage.align = "center";
    - this.overmessage.inputEnabled = true;
    - this.overmessage.events.onInputDown.addOnce(this.quitGame, this);
- Ajouter la fonction suivante
- quitGame: function(pointer) {
- this.state.start('StartMenu');
- },

Test



Adaptation aux mobiles

- Ajoutez les lignes suivantes dans index.html
  - <meta name="mobile-web-app-capable" content="yes">
  - <meta name="viewport" content="width=device-width">
  - <link rel="icon" sizes="196x196" href="bunny.png">

- Pensez à ajouter le fichier bunny.png dans votre projet