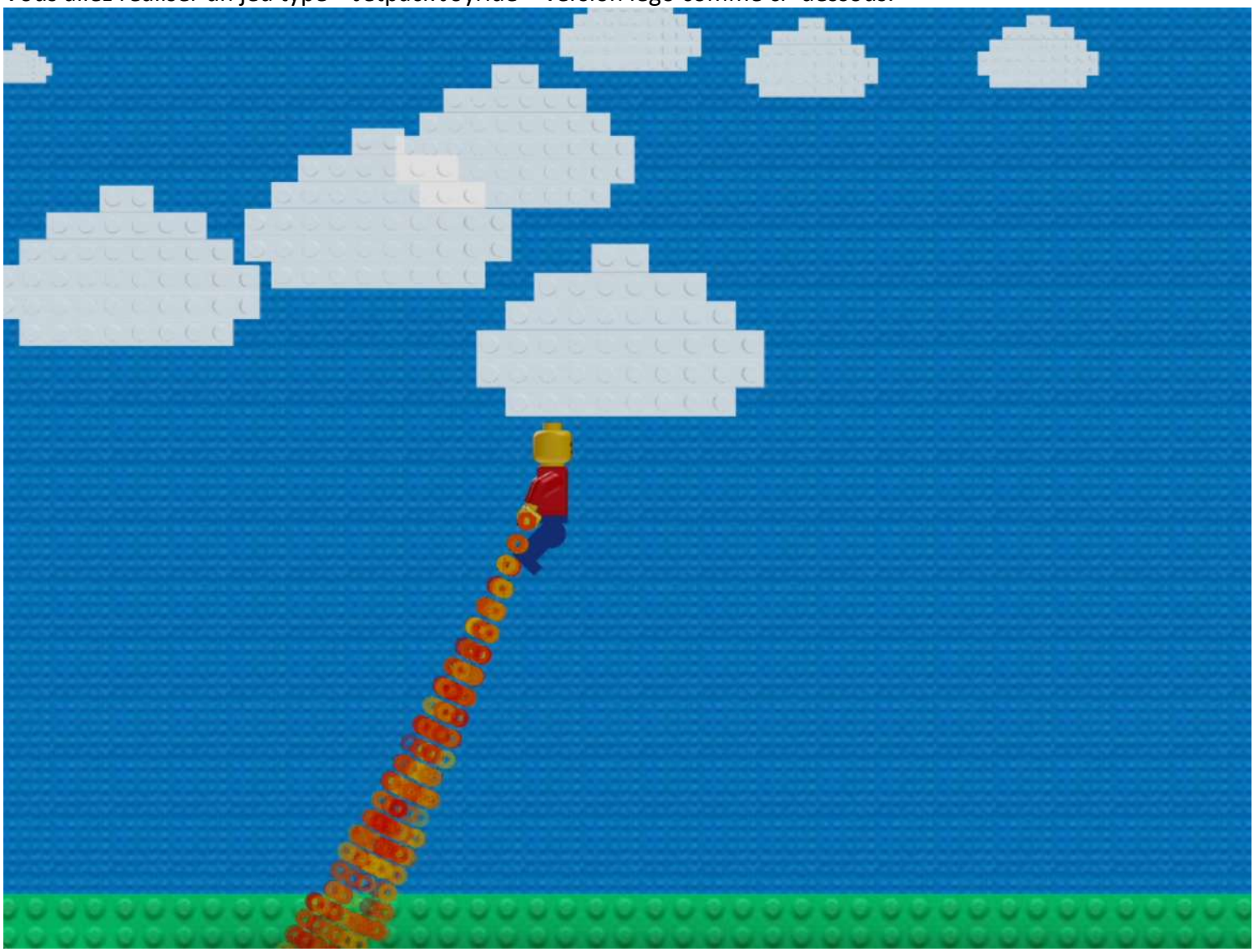
R3.05 – TD3 – 4H

Finir le TD2 avant que commencer le TD3

1 Introduction

Vous allez réaliser un jeu type « Jetpack Joyride » version lego comme ci- dessous.

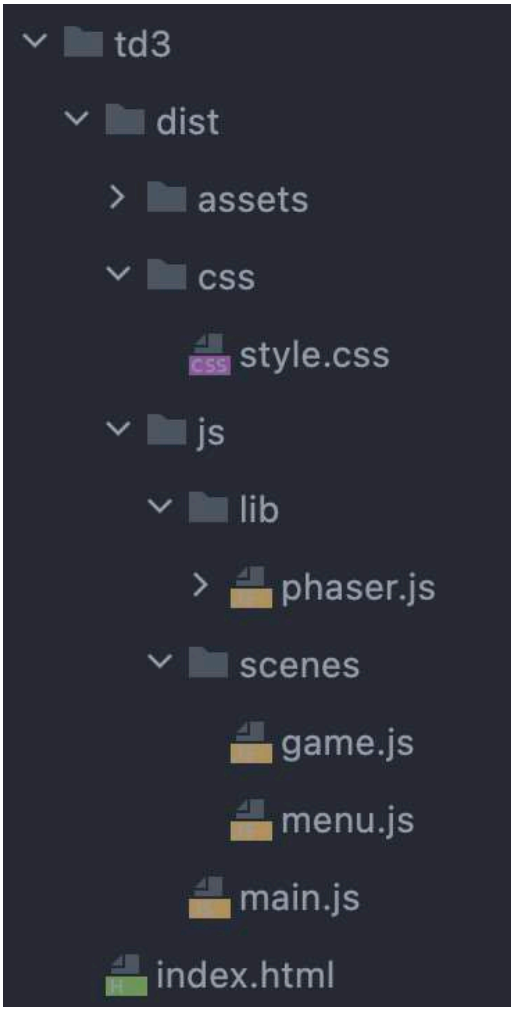


La spécificité de ce td sera au niveau de l'organisation de vos fichiers, « l'architecture » de votre jeu, avec l'ajout de particules et un effet de parallaxe avec les différents éléments du décor.

Le but du jeu est de maintenir la barre espace pour s'envoler, et récupérer des pièces, tuer des ennemis grâce aux particules de feu, et éviter des obstacles.

2 Les fichiers

La première étape avant de créer le jeu va être de le séparer en 3 scènes, un menu, le jeu, et un écran de fin de partie. Pour faire cela, chaque scène va être dans un fichier à part, nous auront donc ces fichiers :



Tous les fichiers qui servent au site sont dans un dossier dist, dans ce dossier on retrouve :

- Un dossier assets dans lequel on y retrouve les images / sons du jeu
- Un dossier css dans lequel on y met notre main.css
- Un dossier js dans lequel on y retrouve :
    - Le dossier lib, ou vous devez y mettre la librairie phaser, donc ces deux fichiers : phaser.js, phaser.min.js)
      Attention récupérez bien depuis le réseau (gestion des particules différente dans la dernière version de phaser)
    - Un dossier scenes, dans lequel vous allez créer un fichier par scène de jeu.
    - Le fichier main.js, qui gèrera les scènes et sera inclus dans l'html.

Enfin on retrouve bien évidemment à la racine de notre projet notre index.html.

Etape 1 :
En suivant la capture d'écran ci-dessus, créez les fichiers et dossiers nécessaires pour que tout soit identique.

Etape 2 : index.html :
On va maintenant inclure dans notre html les différents fichiers, le style, la librairie, et le main de notre jeu :
(Pour les js, il est important d'inclure la librairie puis notre jeu dans le bon ordre, il est également important de dire que notre jeu et de type « module », ça permettra d'inclure nos scènes dans notre main.js

index.html

```html
<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>TD3 : Archi Phaser</title>
    <link rel="stylesheet" href="./dist/css/style.css">
</head>
<body>
<script src="./dist/js/lib/phaser.min.js"></script>
<script type="module" src="./dist/js/main.js"></script>
</body>
</html>
```

3 Gestion des scènes

Étape 3 :
Pour créer une scène il faut créer un fichier (**.js**) dans le dossier js/scenes.
Exemple avec la scène menu, /dist/js/scenes/menu.js

```js
class Menu extends Phaser.Scene {
    constructor() {
        super('Menu');
    }

    preload() {
    }

    create() {
    }

    update(time, delta) {
    }
}

export default Menu;
```

Pour chacune des scènes, il faudra suivre le modèle ci-dessus, voici les étapes à ne pas oublier :

- Nommer la classe par le nom de votre scène (la majuscule est obligatoire)
- Dans le « constructor » , faire : `super('NomDeVotreScene');`
- Créer les 3 fonction preload, create et update (que vous avez déjà utilisé dans les tp précédents)
- Enfin à la toute fin du fichier écrire `export default NomDeVotreScene;` Cette ligne est très importante car elle servira à importer la scène dans le fichier main.js

Étape 4 : main.js :
Il ne nous reste plus qu'à configurer le jeu et importer la scène, tout se fait dans main.js, dans notre cas, notre fichier devrait ressembler à cela :

```js
import Menu from "./scenes/menu.js";
import SceneTest from "./scenes/sceneTest.js";

window.config = {
    width: window.innerWidth / 1.5,
    height: window.innerHeight,
```

```
    mode: Phaser.Scale.FIT,
    autoCenter: Phaser.Scale.CENTER_BOTH,
    type: Phaser.AUTO,
    scene: [Menu, SceneTest],
    backgroundColor: '#2d2d2d',
    physics: {
        default: 'arcade',
        arcade: {
            gravity: {y: 500},
            debug: false
        }
    },
};

const game = new Phaser.Game(window.config);
```

L'import de scènes se fait donc tout en haut du fichier comme ci-dessus.

Important : Pour utiliser une variable d'un fichier à l'autre, elle doit être globale, pour créer une variable globale en javascript, on utilise window.nomDeLaVariable = 'toto' ;
De ce fait, si dans main.js on définit window.config, on va pouvoir accéder à la variable config dans toutes les scènes des différents fichiers.

Remarque : La première scène que Phaser va ouvrir sera la première qui est définie dans la config, comme vous pouvez le voir dans le code ci-dessus.

Étape 5 :
Maintenant que vous savez créer une scène, à vous de répéter l'opération pour créer les scènes Game, et End

4 Passer d'une scène à l'autre

Étape 6 :

Imaginons que vous vouliez créer un bouton « Lancer la partie » dans la scène Menu, vous allez donc le définir dans la fonction create(), de cette façon :

```
this.clickButton = this.add.text(100, 100, 'Start Game', { fill: '#0f0' })
    .setInteractive()
    .on('pointerdown', () => {
        this.scene.start('Game');
    });
```

Vous pouvez donc passer d'une scène à l'autre grâce à l'instruction : `this.scene.start('NomDeLaScene');`

5 Créer le menu

A vous de jouer et laisser libre court à votre imagination ! Créez un menu avec par exemple un fond lego, un texte avec le nom du jeu, et un bouton « Commencer la partie » qui lancera la scène « Game » au click !

6 Créer le jeu

Étape 7 : l'effet parallaxe des nuages :

Comment reproduire l'effet parallaxe des nuages ? En fait c'est tout bête, plus un élément est loin, moins il bouge vite, on va donc créer par exemple 15 nuages, les 5 premiers seront de petite taille, 5 suivants seront de taille moyenne, et enfin les 5 derniers seront à taille réelle.

Pour la création de ceux-ci, on va tout d'abord créer un tableau dans la fonction preload(), celui-ci contiendra tous les nuages, pour qu'on puisse accéder à ceux-ci plus tard dans le jeu.

```
preload() {
    // Le code ici c'est cadeau, mais vous avez déjà vu ceci dans les tp précédents
    this.load.image('sky_23x23', 'dist/assets/img/sky.png');
    this.load.image('grass_23x23', 'dist/assets/img/grass.png');
    this.load.image('piece', 'dist/assets/img/piece.png');
    this.load.image('cloud_300x159', 'dist/assets/img/cloud.png');

    this.clouds = [];
}
```

On va ensuite dans la fonction create(), et on fait une double boucle for, qui va itérer 15 fois au total :

```
// Première boucle qui itère 3 fois, car il y a 3 couches de nuages
for (let couche = 1; couche <= 3; couche++) {

    // Deuxième boucle qui itère 5 fois, car il y a 5 nuages par couche
    for (let t = 0; t < 5; t++) {
        let cloud = this.add.image(
            Math.random() * window.config.width - 300 * 0.2 * couche, // Position x aléatoire
sur la map
            (Math.random() * window.config.height - 159 * 0.2 * couche) / 3, // Position y
aléatoire sur la map
            'cloud_300x159', // Texture du nuage
        )
            .setOrigin(0, 0) // On définit l'origine des transformations au coin en haut à
gauche
            .setScale((Math.random() * 0.05 + 0.05) * couche);
        // on redimensionne le nuage aléatoirement par rapport à sa couche, plus la couche
est grande plus le nuage sera aussi grand

        cloud.alpha = 0.9; // on met le nuage à 90% d'opacité
        this.clouds.push(cloud); // on ajoute le nuage au tableau précédemment créé
    }
}
```

Enfin pour déplacer les nuages à une vitesse correspondant à leur taille, on se rend dans la fonction update() :

```
this.clouds.forEach((cloud) => {
    cloud.x -= (delta * 30) / 100 * cloud.scale;

    // si le nuage se retrouve en dehors de la fenêtre,
    // on le téléporte à droite, pour donner l'impression
    // qu'un autre nuage arrive, alors que c'est le même
    if(cloud.x < 0 - cloud.width) {
        cloud.x = window.config.width;
    }
});
```

Bravo ! Vous avez des nuages qui se déplacent sur la carte de droite à gauche, plus ou moins gros, plus ou moins vite !

Étape 8 : Créer un ciel :

Remarque : à placer avant les nuages, sinon ceux-ci se retrouverons derrière le ciel et on ne les verra plus.

Dans la fonction preload(), créer une variable sky : `this.sky;`
Puis charger la texture du ciel :

```
this.load.image('sky_23x23', 'dist/assets/img/sky.png');
```

Dans la function create() ajouter le ciel :

```
this.sky = this.add.tileSprite(0, 0, window.config.width/2, window.config.height/2,
"sky_23x23").setOrigin(0, 0).setScale(2),
```

Dans la fonction update() modifier la position x de la texture en fonction du temps écoulé pour donner l'impression qu'il se déplace de droite à gauche : `this.sky.tilePositionX = time / 20;`

Étape 9 : Créer le sol :
En vous basant sur le ciel créé ci-dessus, à vous de jouer et d'ajouter un sol (grass.png) qui se déplace assez rapidement (car le sol est l'élément le plus proche, donc par rapport à la parallaxe il se déplace rapidement).

Etape 10 : Créer le personnage :
En vous basant sur les tp précédents, créez un personnage qui sera contrôlable (spritesheet.png), il n'aura que 3 animations :
running (start: 0, end: 3, framerate: 10)
falling (start: 4, end: 5, framerate: 5)
jumping (start: 7, end: 6, framerate: 10, repeat: 0)

N'oubliez pas d'ajouter les collisions entre le joueur, et le sol :

```
this.physics.add.existing(this.grass, true);
this.physics.add.collider(this.player, this.grass);
```

Etape 11 : Contrôler le personnage :

La logique est simple : quand on clique sur espace, on diminue la vélocité en y du personnage pour le faire voler et on lui applique l'animation « jumping »

Sinon s'il touche le sol on lui applique l'animation « running », et s'il touche le sol et qu'il est orienté vers la gauche, le joueur va donc « suivre » le sol, et on va lui appliquer la vitesse du sol

Enfin si on clique à gauche ou droite, on utilise .flipX pour retourner le joueur sans à avoir à créer des animations différentes pour qu'il aille à gauche ou à droite, et on le déplace sur l'axe X.

```
// Si on appuie sur "espace"
if (this.cursors.space.isDown || this.input.activePointer.isDown ) {
    this.player.body.velocity.y -= 20
    this.player.anims.play('jumping', false);
} else if (this.player.body.touching.down) {
    if(this.player.flipX === true) {
        this.player.x -= 0.5 * delta;
```

```
    }
    this.player.anims.play('running', true);
} else {
    if (this.player.body.velocity.y > 10) {
        this.player.anims.play('falling', true);
    } else {
        this.player.anims.play('jumping', false);
    }
}

// Si on clique sur 'gauche' ou 'droite'
if (this.cursors.left.isDown) {
    this.player.flipX = true;
    this.player.x -= 0.3 * delta;
} else if(this.cursors.right.isDown) {
    this.player.flipX = false;
    this.player.x += 0.3 * delta;
}
```

Etape 12 : Les particules de feu :

Le but de cette étape va être de faire apparaître des particules de feu derrière le personnage lorsqu'il s'envole. Pour cela on va tout d'abord créer une variable dans le preload : `this.particles;`

On va ensuite créer les particules dans le create :

```
this.particles = this.add.particles('piece').createEmitter({
    speed: 1000,
    scale: { start: 0.05, end: 0.05 },
    lifespan: 1000,
    angle: { min: 110, max: 120 },
    quantity: {min: 10, max: 50},
    tint: [ 0xcc0000, 0xcc5a00, 0xccab00 ],
    on: false,
});
```

Et on dit que la position des particules suit celle du joueur (avec un décalage de -20px et 15px pour qu'elles soient au niveau des mains du joueur) :

```
this.particles.startFollow(this.player, -20, 15);
```

Il faut ensuite faire apparaître / disparaître les particules aux bons moments, on reprend donc le code de l'étape 11 pour y ajouter la gestion des particules :

```
if (this.cursors.space.isDown || this.input.activePointer.isDown ) {
    // ...
    this.particles.start()
} else if (this.player.body.touching.down) {
    // ...
    this.particles.stop();
} else {
    // ..
}

if (this.cursors.left.isDown) {
    // ...
    this.particles.setAngle( { min: 30, max: 40 });
    this.particles.startFollow(this.player, 20, 15);
} else if(this.cursors.right.isDown) {
    // ...
    this.particles.setAngle( { min: 110, max: 120 });
    this.particles.startFollow(this.player, -20, 15);
}
```

Etape 11 : Les effets sonores :

C'est bien connu, on n'a pas de bon jeu sans bon sons, pour cela vous disposez de 10 sons, il faut tout d'abord les charger dans le preload :

```
this.load.audio('step_left_1', 'dist/assets/sounds/boots_step_left_1.wav');
this.load.audio('step_left_2', 'dist/assets/sounds/boots_step_left_2.wav');
this.load.audio('step_left_3', 'dist/assets/sounds/boots_step_left_3.wav');

this.load.audio('music', 'dist/assets/sounds/music.mp3');
// à vous de faire le reste
```

Puis les ajouter dans une variable dans le create :

```
this.sounds = {
    step_left: [
        this.sound.add('step_left_1'),
        this.sound.add('step_left_2'),
            this.sound.add('step_left_3'),
        ],
        music: this.sound.add('music', { volume: 0.5 }),
    // à vous d'ajouter tous les autres
}
```

En utilisant cette syntaxe, vous pouvez lancer la musique en faisant : `this.sounds.music.play();`
Vous pouvez jouer un son de pas de cette façon : `this.sounds.step_left[0].play();`

Le but est de :

- Jouer la musique au début de partie, et en boucle
- Jouer un son de pas une fois sur deux (une fois step_left, une fois step_right) et au hasard
  Par exemple un son step_left au hasard parmis les 3 :
  ```
  const rand = Math.floor(Math.random() * (3));
  this.sounds.step_right[rand].play();
  ```
- Jouer un son lorsque le personnage saute (boots_jump.wav)
- Jouer un son lorsque le personnage atterri (boots_land.wav)
- Jouer un son lorsque les particules de feu sont affichées (flying.wav)

7 Pour les plus rapides : Les pièces et obstacles

A vous de jouer, en vous aidant du tp 1 et 2, créer des pièces qui apparaitrons aléatoirement dans le ciel et qui viendront augmenter votre score de 1 point lorsqu'elles seront récupérées.
Remarques :

- Une pièce se déplace aussi de droite à gauche, vous pouvez utiliser la même logique que pour les nuages pour les téléporter à droite lorsqu'elles ne sont plus visibles à gauche
- Une pièce est au premier plan, donc par rapport à la parallaxe, elle va à la même vitesse que le sol
- Vous avez une image disponible pour la pièce (ai-coin.png). (Petite info : cette image n'existe nulle part ailleurs et a complètement été créée par intelligence artificielle)
- Conseil : lorsque vous ramassez une piece, votre score augmente de 1 point, stockez votre score dans une variable globale (par ex : window.score = 0 ; puis window.score ++ ;), car vous aurez besoin de celui-ci dans la scène de mort (End) ou par exemple dans la scène Menu avec l'affichage du dernier score.
- Si vous trouvez ce tp trop simple, faites un historique des scores que vous enregistrez dans les cookies, et que vous pourrez afficher sur la scène Menu.

Toujours grâce au tp 1 et 2, créez des obstacles, ceux-ci causeront la mort du personnage lorsqu'il rentrera en contact avec eux (vous pourrez donc depuis ici lancer la scène « End », qui affichera votre score).

Les obstacles seront des arbres ou des rochers (tree.png et rock.png). Ces obstacles se déplacent aussi à la vitesse du sol, et sont positionnés sur le sol, donc évitez de trop en faire apparaître si vous voulez réussir à faire un bon score !