

# TP Déploiement de Service

réalisation d'une application en environnement collaboratif

<b>1.Présentation de la ressource.....</b>	<b>2</b>
<b>2.Notions de bases.....</b>	<b>2</b>
2.1. npm.....	2
2.1.a. Qu'est ce que npm.....	2
2.1.b. Installation de npm.....	3
2.1.c. Utilisation de npm.....	4
2.1.c.1. Installation de package.....	4
2.1.c.2. Commande de build, de test .....	4
2.2. Versionning.....	4
2.2.a.Définition.....	4
2.2.b. Versioning sémantique.....	5
2.2.c. GIT.....	5
2.3. Webpack.....	5
<b>3.Création de la V1 de l'application space invader.....</b>	<b>7</b>
3.1.préparation de l'environnement de travail.....	7
3.1.a.Créer un github.....	7
3.1.b.NPM.....	7
3.1.c.VSCode.....	8
3.2.Structure du projet.....	8
3.3.conception de la V1 du jeu.....	9
3.3.a.Structure du projet.....	9
3.3.b.Stylisation de base.....	10
3.3.c.Préparation du canvas.....	10
3.3.d.mise en place de la gameloop.....	12
3.3.e. Graphisme du jeu- affichage du background.....	13
3.3.f. Affichage des invaders.....	13
3.3.f.1. création de la classe Invader.....	15
3.3.f.2.Affichage des invaders.....	17
3.3.g. Mouvement des invaders.....	18
3.3.h. Affichage du Player.....	22
3.3.i.Contrôle du player.....	23
3.3.j. tir du vaisseau.....	25
3.3.h. tir des invaders.....	29
3.3.i.Collision bullets x invaders.....	30
3.3.j. GameOver.....	32
3.3.j.1.une bullets touche le player.....	32
3.3.j.2.un invader touche le player.....	33
3.3.j.3.tous les invaders sont éliminés.....	35
3.3.j.4.afficher le texte de fin de jeu.....	35
<b>4.nouvelles features dans les autres versions.....</b>	<b>37</b>

V1.1 ajout d'un bouton replay.....	37
V1.2 ajout du score.....	37
V1.3 ajout de nouvelle map invader.....	38
V1.4 ajout de niveau de difficulté.....	38

# 1.Présentation de la ressource

Dans cette ressource vous allez:

- concevoir un jeu en javascript
- travailler dans un environnement collaboratif: git, création de branche
- utiliser des gestionnaires de paquets
- gérer le versionning de votre application
- créer des script pour le développement, le build et le déploiement de vos applications

Déroulé des 8H de TP:

- séance 1 :
  - explication des notions de bases
  - mise en place de l'environnement ( visual studio, npm)
  - début du dev du space invader V1.0.0
- séance 2:
  - fin de la V1.0.0 - création d'une nouvelle branche
  - script de build et déploiement sur localhost
- séance 3:
  - début de la V1.1.0
  - code review
  - pris en compte des remarques du code review et commit de la V1.1.1
- séance 4:
  - travail sur les nouvelles feature, V1.2.0, V1.3.0
  - QCM

## 2.Notions de bases

### 2.1. npm

#### 2.1.a. Qu'est ce que npm

npm est l'abréviation de Node Package Manager. C'est un gestionnaire de paquet. Il permet d'accéder à un ensemble de bibliothèques & frameworks javascript qui vont vous servir pour votre propre projet.

Par exemple jQuery, Bootstrap ou react sont des packages.

Pour simplifier, un package, c'est du code que vous n'avez pas écrit mais que vous avez obtenu d'une source publique pour l'utiliser dans votre projet.

Un paquet est du code que vous n'avez pas écrit mais que vous avez obtenu d'une source publique pour l'utiliser dans votre projet.

Vous avez déjà probablement utilisé un package

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
```

```
<script src="/jquery.min.js"></script>
```

La simple liaison à un paquet dans une balise de script fonctionne bien, tant que :

- le nombre de projets que vous avez est gérable ;
- le nombre de personnes travaillant sur les projets est gérable ;
- le nombre de mises à jour qui doivent être apportées aux paquets est gérable ;
- Chaque paquet utilisé dans vos projets est du côté client (navigateur) JavaScript ou CSS.

Un gestionnaire de paquet est un autre moyen d'accéder à vos packages.

### 2.1.b. Installation de npm

npm est livré avec node. Pour installer npm nous allons donc installer nvm qui permet de gérer les version de node.

Avant d'installer npm, nous devons vérifier s'il est déjà installé. Pour cela vous devez ouvrir un terminal et lancer la commande suivante:

```
node - v
```

Cette commande affiche la version actuelle de Node.. Si Node est installé, la ligne de commande répondra avec le numéro de version de Node qui est actuellement installé :

```
v20.11.0
```

Si aucune version n'est installée sur votre poste, suivez les instructions suivantes :

- Téléchargez la dernière version de NVM pour Windows. Vous pouvez l'installer manuellement, si vous préférez.
- Ouvrez le terminal et exécutez la commande "nvm list available" pour voir une liste des versions de Node qui sont disponibles pour le téléchargement et l'installation.
- Exécutez la commande nvm use, suivie du numéro de version de Node que vous souhaitez utiliser (par exemple, nvm use 16.9.1) pour utiliser une version spécifique. Vous pouvez également utiliser use latest, lts, ou newest au lieu d'un numéro de version spécifique, où newest est la dernière version installée.

### 2.1.c. Utilisation de npm

Npm s'utilise en ligne de commande. Il vous faut donc un terminal.

#### 2.1.c.1. Installation de package

pour installer un package on peut utiliser deux commandes:

- **npm install** (ou **npm i**) est la manière standard (et par défaut) d'ajouter un paquet à un projet.
- **npm install --save-dev** (ou **npm i -D**) ajoute uniquement le paquet à vos "dépendances de développement", ce qui signifie qu'il ne sera installé que lors du développement du projet, et non lors de la construction de la version de production finalisée du projet.

Cette deuxième méthode peut être utile pour installer des bibliothèque de tests, des linter ...

Lorsque l'on installe un paquet avec NPM, celui-ci apparaît dans un fichier `package.json`. Le fichier `package.json` définit les dépendances du projet et ses métadonnées.

### 2.1.c.2. Commande de build, de test ...

npm permet également de créer des script appelables avec des commandes. Ces scripts sont définis dans le fichier `package.json`.

Dans un projet node, on peut ainsi définir par exemple des scripts pour:

- lancer le projet sur un live server
- lancer le build du projet
- lancer les tests du projet
- déployer le projet

## 2.2. Versionning

### 2.2.a.Définition

Une version d'un logiciel correspond à un état donné de l'évolution d'un produit logiciel utilisant le versionnage.

Le versionning sert pour:

- développer des fonctionnalités en parallèle,
- tester des changements dans des branches séparées,
- fusionner les changements dans la branche principale sans perturber le flux de travail.

Il existe différentes méthodes pour faire du versionning:

- Date de sortie, par exemple Ubuntu 18.04 est sortie en avril 2018
- Numérotation sémantique
- autres : Codes alphanumériques, Numerotation unaire..

### 2.2.b. Versioning sémantique

Le **versioning sémantique** est un exemple populaire qui utilise un format MAJEUR.MINEUR.CORRECTIF pour indiquer l'impact des changements. Cette clarté dans la version aide les développeurs à comprendre l'ampleur des modifications apportées. par exemple pour la sortie d'un jeu:

- la première version commercialisable sera la version 1.0.0
- si le jeu connaît quelques bugs, le jeu avec les correctifs sera tagué en version 1.0.1, 1.0.2, etc ...
- si on souhaite ajouter une nouvelle fonctionnalité, comme un nouveau mode de jeu alors il s'agit d'une modification mineur, que l'on pourra taguer version 1.1.0

Une nouvelle version doit toujours être accompagnée d'une release note qui indique les changements apportés à l'application.

### 2.2.c. GIT

Git est un système de **gestion de versions** partagé qui fut développé en 2005 par le créateur de Linux Linus Thorvalds et publié sous licence libre GNU-GPLv2. Même si chaque projet dispose d'un répertoire central, la particularité de cet outil réside dans le fait que tous les utilisateurs participants téléchargent une copie de travail locale de ce répertoire sur leur appareil.

Chacune de ces copies constitue une sauvegarde à part entière du répertoire, ce qui implique qu'une connexion permanente au réseau n'est pas nécessaire pour le processus de travail sous-jacent.

## 2.3. Webpack

Webpack est un bundler JavaScript pour vos applications Web. il permet de compiler le code html et javascript dans un seul et même fichier à déployer ensuite sur votre serveur.

Pour fonctionner correctement le webpack nécessite une structure de projet défini qui doit être respectée.

Nous utiliserons Webpack pour notre projet space invader.

Nous verrons notamment

- comment importer des assets : son et images
- comment lancer un live server
- comment faire un build

Le fonctionnement de webpack pour votre projet est défini dans le fichier webpack.config.js. En voici un exemple

You, 18 hours ago | 1 author (You)

```
const path = require('path');
```

```
module.exports = {  
  // 1  
  entry: './src/index.js',  
  // 2  
  output: {  
    path: path.resolve(__dirname, 'dist'),  
    filename: 'bundle.js'  
  },  
  // 3  
  devServer: {  
    static: './dist'  
  },  
  module: {  
    rules: [  
      {  
        test: /\.scss|css$/,  
        use: ['style-loader', 'css-loader', 'sass-loader'],  
      },  
      {  
        test: /\.(png|svg|jpg|jpeg|gif)$/i,  
        type: 'asset/resource',  
      },  
      {  
        test: /\.wav$/,  
        loader: 'file-loader',  
      }  
    ]  
  },  
};
```

You, 23 hours ago • 3.3.f.affichage des invaders

## 3.Création de la V1 de l'application space invader

### 3.1.préparation de l'environnement de travail

#### 3.1.a.Créer un github

Prérequis : git doit être installé sur votre PC.

- Créez un nouveau repo git sur github.com
- Dans le terminal à la racine de votre projet, faites un "git clone urldeVotreProjet"
- Créer un nouveau dossier spaceInvader
- à la racine du dossier créer un fichier .gitignore (il sera vide pour le moment)
- dans votre projet, ajouter le dossier "assets" issu du zip dispo sur moodle

#### 3.1.b.NPM

Prérequis : npm doit être installé sur vos postes

- commencez par lancer la commande `npm init` pour initier un nouveau projet
- installez sass : `npm install -g sass`
- `npm install --save-dev sass-loader node-sass`
- installez webpack: `npm install --save-dev webpack webpack-dev-server webpack-cli`

Nous allons créer un premier script qui permet de lancer un live server pour notre projet via webpack.

Dans le fichier package.json, ajouter la commande suivante:

```
{
  ...
  "scripts": {
    "start": "webpack serve --config ./webpack.config.js --mode development"
  },
  "keywords": [],
  ...
}
```

Pour lancer le live server il suffira de faire la commande `npm run start`

Créer un fichier webpack.config.js dans lequel vous préciserez:

```
module.exports = {
  // 1
  entry: './src/index.js',
  // 2
  output: {
```



```
    path: '/dist',
    filename: 'bundle.js'
  },
  // 3
  devServer: {
    static: './dist'
  }
};
```

Une fois les dépendances installées, renseigner le fichier gitignore

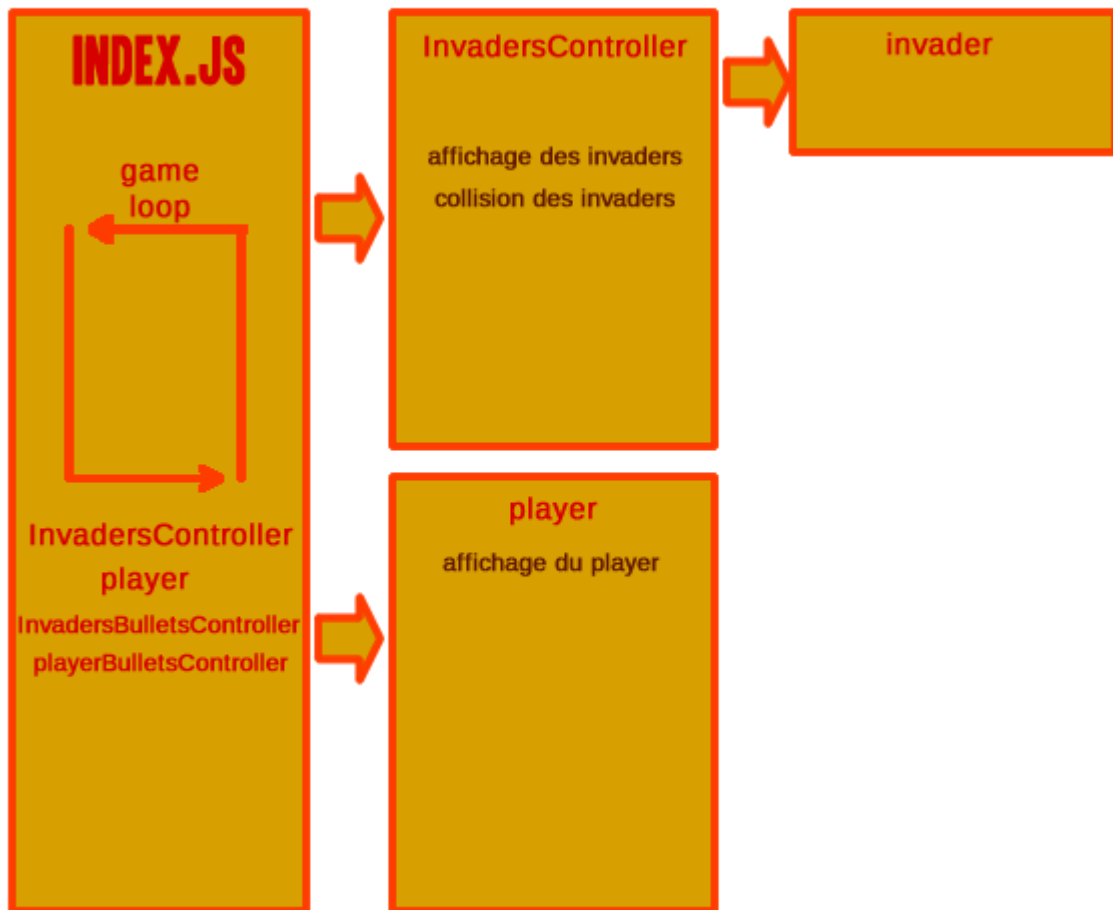
```
node_modules/*
```

et faites un premier commit de votre projet: `git commit -m "initial commit"`

## 3.2. Structure du projet

- un dossier "dist"
- un dossier "src"
- un dossier "assets" (avec les assets disponible sur moodle) dans le dossier "dist"
- un fichier "index.js" dans le dossier src
- un fichier "index.html" dans le dossier "dist", ce fichier doit charger le fichier bundle.js  
`<script src="./bundle.js"></script>`
- un fichier package.json à la racine du projet

### 3.3.conception de la V1 du jeu



#### 3.3.a.Structure du projet

créer une nouvelle branche “develop” qui sera votre branche pour les développements en cours.

```
git branch develop
```

puis positionnez vous sur cette branche:

```
git checkout develop
```

Dans index.html:

- créer la structure de base d’une page html avec comme titre de page le nom de votre jeu, par exemple “the most amazing Space Invader”
- ajouter un canvas avec l’id “game”

Tester la page en lançant le serveur webpack : npm start

### 3.3.b.Stylisation de base

- créez un fichier styles.scss dans un dossier css
- ajouter un style au canvas pour le détacher du fond
- styliser également votre titre
- styliser votre body

pour que webpack traite votre scss , modifiez le webpack.config.js en ajoutant:

```
module: {
  rules: [
    {
      test: /\.scss$/,
      use: ['style-loader', 'css-loader', 'sass-loader'],
    }
  ]
},
```

puis ajoutez l'import du css dans le fichier src/index.js

```
import '../css/style.scss';
```

Pour lancer la compilation faites `npm run start`.

**Attention** : Si à cette étape vous avez des erreurs, c'est que probablement il vous faut installer des loaders pour webpack.

```
npm install --save-dev style-loader css-loader
```

### 3.3.c.Préparation du canvas

Dans `index.js`:

- récupérer le canvas dans une variable canvas.
- définissez le contexte de ce canvas comme 2D et récupérer le contexte dans une variable que vous nommerez ctx.
- donnez au canvas une taille de 600px par 600px

```
let canvas = document.getElementById("game");
let ctx = canvas.getContext("2d");
canvas.height=600;
canvas.width=600;
```

documentation

<https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D>

Toujours dans le fichier **index.js**:

- créer une image dans une variable **background**.
- adresser lui comme source l'image **space.png**.

```
const background = new Image();
background.src = "space.png";
```

webpack permet de gérer le chargement d'images avec des imports.  
pour cela il faut installer un file loader:

```
npm install --save-dev file-loader
```

puis modifier le fichier **webpack.config.js** pour qu'il sache gérer le type de fichier.

```
module: {
  rules: [
    {
      test: /\.scss$/,
      use: ['style-loader', 'css-loader', 'sass-loader'],
    },
    {
      test: /\.(png|svg|jpg|jpeg|gif)$/i,
      use: [
        'file-loader',
      ]
    }
  ]
},
```

enfin faire un import dans le fichier **index.js**

```
import Space from './assets/images/space.png';
```

à cette étape, votre jeu devrait ressembler à cela

the most amazing Space Invader

### 3.3.d.mise en place de la gameloop

#### définition gameloop

La gameLoop ou boucle de jeu est souvent appelée une 'boucle' car le programme répète certaines actions jusqu'à ce qu'il soit arrêté manuellement.

Elle englobe les processus essentiels qui se produisent pendant que le jeu est en cours d'exécution.

Ces processus incluent :

- Entrée : Comment le joueur interagit avec le jeu.
- Mise à jour : Gestion de la logique du jeu, de la physique et d'autres changements dynamiques.
- Rendu : Affichage des visuels du jeu à l'écran.

Pour mettre en place une gameLoop il faut que vous créiez une fonction qui est appelée 60 fois par secondes.

Dans le fichier **index.js** :

- créer une fonction game qui sera vide pour le moment.
- faire en sorte que cette fonction soit appelé 60 fois par seconde

```
function game(){  
  
}  
  
setInterval(game, 1000/60);
```

### 3.3.e. Graphisme du jeu - affichage du background

Dans la fonction game, demandé au contexte du canvas d'afficher l'image de background que vous avez préalablement défini.

```
function game(){  
    ctx.drawImage(██████████, 0,0, canvas.width, canvas.height);  
}
```

documentation

<https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/drawImage>

### 3.3.f. Affichage des invaders

Créez un nouveau fichier **InvaderController.js** dans le dossier src. Ce fichier contiendra une classe InvaderController.

- faites en sorte que ce fichier soit importable dans le fichier index.js
- le fichier index.js fera un import de ce module pour instancier la classe invaderController.

La classe InvaderController doit contenir :

- un constructor de classe pour instancier la classe
- une fonction draw qui prend en paramètre le contexte du canvas et qui sera chargé de dessiner les invaders.

Dans la classe InvaderController, Définissez le constructor de la classe qui prend en paramètre un canvas et définit le canvas de la classe avec ce canvas passer en paramètre.

Pour l'affichage des invaders, vous utiliserez un tableau de tableau (invadersMap) qui représente la position de chaque invader.

```
invadersMap = [  
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],  
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],  
    [2, 2, 2, 3, 3, 3, 3, 2, 2, 2],  
    [2, 2, 2, 3, 3, 3, 3, 2, 2, 2],  
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],  
    [2, 2, 2, 2, 2, 2, 2, 2, 2, 2],  
];
```



Dans `invadersMap`, chaque numéro (1,2 & 3) correspond à une image disponible dans le dossier `images`.

```
export default class InvaderController {

  invadersMap = [
    [1,1,1,1,1,1,1,1,1,1],
    [1,1,1,1,1,1,1,1,1,1],
    [1,1,2,2,3,3,2,2,1,1],
    [1,1,1,2,1,1,2,1,1,1],
    [1,1,1,2,1,1,2,1,1,1],
    [1,1,1,1,1,1,1,1,1,1]
  ]

  constructor(canvas) {
    this.canvas = canvas;
  }

  draw(ctx){

  }

}
```

toujours dans **InvaderController.js**:

- créer un tableau `invadersRows` qui servira à stocker les lignes d'invaders
- créer ensuite une fonction `createInvaders()` qui servira à dessiner les invaders sur le `canva`.
- Cette fonction doit être appelée depuis le constructeur de la classe.
- dans cette fonction vous allez parcourir le tableau `invaders map` afin de créer les bon invaders aux bons endroits :

```

createInvaders() {
  this.invadersMap.forEach((row, rowIndex) => {
    this.invaderRows[rowIndex] = [];
    row.forEach((invaderNubmer, invaderIndex) => {
      if (invaderNubmer > 0) {
        this.invaderRows[rowIndex].push(
          new Invader(invaderIndex * 50, rowIndex * 35, invaderNubmer)
        );
      }
    });
  });
}

```

Dans l'état actuel, la classe Invader n'existe pas.

### 3.3.f.1. création de la classe Invader

pour afficher un invader, nous allons devoir afficher une image.

#### comment importer des images avec webpack

il faut ajouter une règle au fichier webpack.config.js

```

module: {
  rules: [
    {
      test: /\.scss$/,
      use: ['style-loader', 'css-loader', 'sass-loader'],
    },
    {
      test: /\.png|svg|jpg|jpeg|gif$/i,
      type: 'asset/resource',
    },
  ],
}

```

puis ensuite faire l'import du fichier comme pour un fichier JS dans le fichier qui en a besoin:

```
import Invader1 from '../assets/images/invader1.png';
```

Nous allons maintenant créer la classe Invader qui permettra d'instancier des invader. chaque invader a:

- une variable x: position en abscisse
- une variable y: position en ordonné
- une variable width: sa largeur
- une variable height : sa hauteur
- une variable image: un élément html image dont la source doit être une image invader du dossier image.



```

import Invader1 from '../assets/images/invader1.png';
import Invader2 from '../assets/images/invader2.png';
import Invader3 from '../assets/images/invader3.png';

export default class Invader {
  constructor(x, y, imageNumber) {
    this.x = x;
    this.y = y;
    this.width = 44;
    this.height = 32;

    this.image = new Image();
    this.image.src = getImage(imageNumber);
  }

  getImage(imageNumber) {
    switch (imageNumber) {
      case 1:
        return Invader1;
      case 2:
        return Invader2;
      case 3:
        return Invader3;
      default:
        return Invader1;
    }
  }
}

```

Les invaders doivent pouvoir être affichés à l'écran. Pour cela vous allez **créer une méthode draw dans la class invader**.

Cette méthode devra utiliser le contexte du canvas pour afficher l'image de l'invader.

```

draw(ctx) {
  ctx.drawImage( , , , , );
}

```

Faites un import de classe Invader dans InvaderController.

### 3.3.f.2. Affichage des invaders

#### Dans InvaderController:

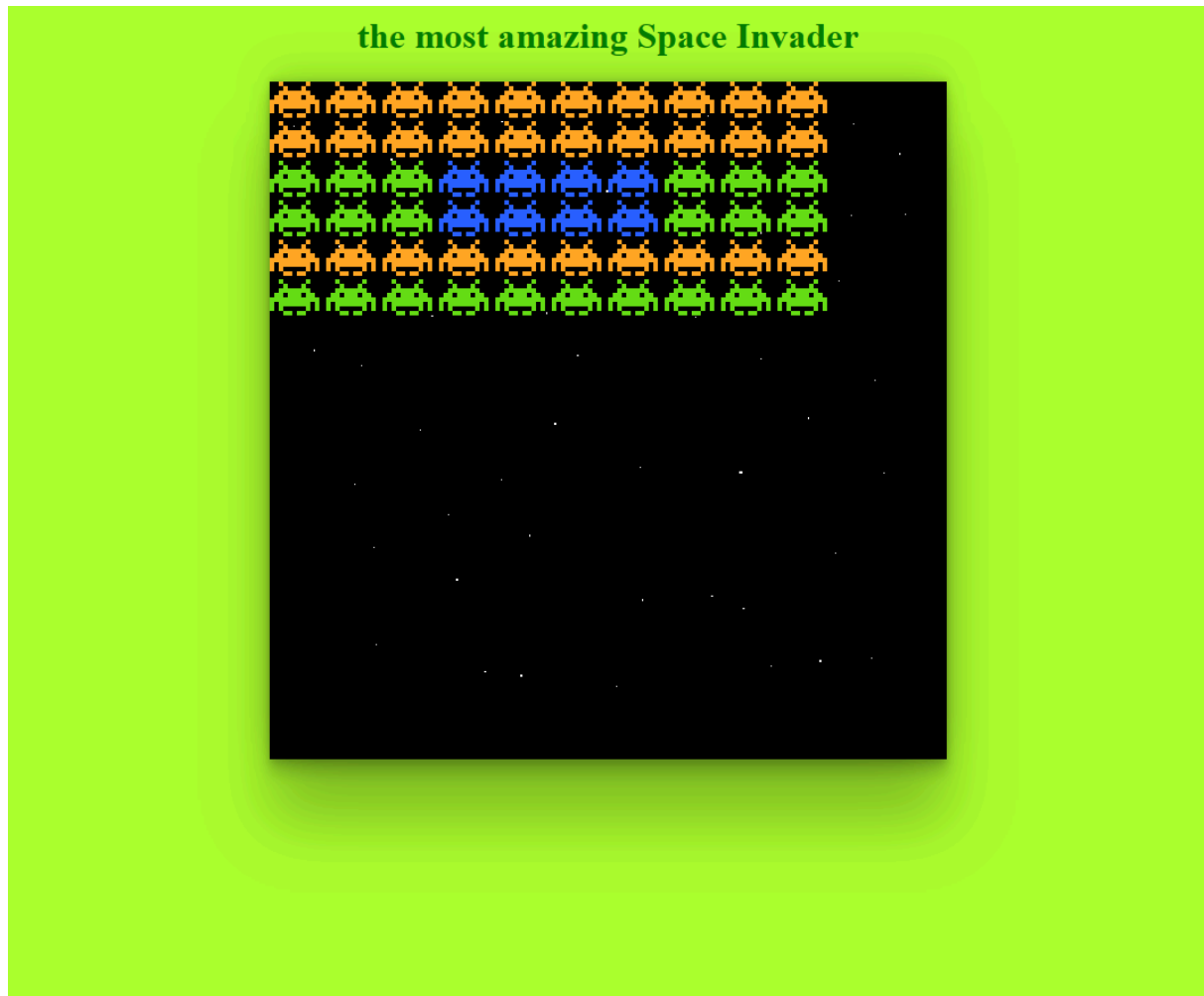
- ajouter une fonction draw qui permettra de dessiner les invaders.
- créer une fonction drawInvaders qui sera appelé depuis la fonction draw
- la fonction drawInvaders fait appel à la fonction draw de chaque invader.

```
drawInvaders(ctx){  
  this.invaderRows.flat().foreach((invader)=>{  
    invader.draw(ctx);  
  })  
}
```

Il vous reste maintenant à importer la classe invaderController dans index.js.  
puis dans index.js:

- importer la classe invaderController
- créer une instance de invaderController
- faire un appel à la fonction draw de invaderController dans la gameLoop.

à cette étape vous devriez avoir un résultat de ce type



### Commit et push

Si c'est le cas faites un commit et pusher sur votre branche develop. Avec comme message "3.3.f.affichage des invaders"

```
git commit -a -m "3.3.f.affichage des invaders"
```

### 3.3.g. Mouvement des invaders

Créez un nouveau fichier **movingDirection.js** dans le répertoire src.

Définissez y une variable constante de type objet qui définit les différentes directions possibles des invaders.

Chaque direction est un nombre pour que ...?

```
const MovingDirection = {  
  left: 0,  
  right: 1,  
  downLeft: 2,  
  downRight: 3,  
};  
  
export default MovingDirection;
```

### définition

La vitesse combine la vitesse à laquelle un objet se déplace avec la direction dans laquelle il se déplace à un moment donné

Dans **InvaderController**:

- Importez le module MovingDirection.
- Ajoutez des variables :
  - currentDirection qui prend pour valeur MovingDirection.right
  - xVelocity initié à 0;
  - yVelocity initié à 0;
  - defaultXVelocity = 1;
  - defaultYVelocity = 1;
- Créez ensuite une fonction updateVelocityAndDirection qui doit
  - mettre à jour la vitesse et la direction des invaders.
  - être appelé depuis la fonction draw

```

updateVelocityAndDirection() {
  for (const invaderRow of this.invaderRows) {
    if (this.currentDirection === MovingDirection.right) {
      this.xVelocity = this.defaultXVelocity;
      this.yVelocity = 0;
      const rightMostInvader = invaderRow[invaderRow.length - 1];
      if (rightMostInvader.x + rightMostInvader.width >= this.canvas.width) {
        this.currentDirection = MovingDirection.downLeft;
        break;
      }
    } else if (this.currentDirection === MovingDirection.downLeft) {
      if (this.moveDown(MovingDirection.left)) {
        break;
      }
    } else if (this.currentDirection === MovingDirection.left) {
      this.xVelocity = -this.defaultXVelocity;
      this.yVelocity = 0;
      const leftMostInvader = invaderRow[0];
      if (leftMostInvader.x <= 0) {
        this.currentDirection = MovingDirection.downRight;
        break;
      }
    } else if (this.currentDirection === MovingDirection.downRight) {
      if (this.moveDown(MovingDirection.right)) {
        break;
      }
    }
  }
}

```

updateVelocityAndDirection fait appel à une autre fonction moveDown qui a pour but de faire descendre les invaders d'une ligne.

```

moveDown(newDirection) {
  this.xVelocity = 0;
  this.yVelocity = this.defaultYVelocity;
  if (this.moveDownTimer <= 0) {
    this.currentDirection = newDirection;
    return true;
  }
  return false;
}

```

Cette méthode fait appel à une variable moveDownTimer définie au niveau global. Cette variable est décrémentée lorsque les invaders font un mouvement vers le bas.

```

moveDownTimerDefault = 30;
moveDownTimer = this.moveDownTimerDefault;

```

Pour décrémenter le timer nous faisons appel à des fonctions qui sont appelées depuis la fonction draw.

```
resetMoveDownTimer() {
    if (this.moveDownTimer <= 0) {
        this.moveDownTimer = this.moveDownTimerDefault;
    }
}

decrementMoveDownTimer() {
    if (
        this.currentDirection === MovingDirection.downLeft ||
        this.currentDirection === MovingDirection.downRight
    ) {
        this.moveDownTimer--;
    }
}
```

Ces deux fonctions doivent être appelées depuis la fonction draw du invader controller.

```
draw(ctx){
    this.decrementMoveDownTimer();
    this.drawInvaders(ctx);
    this.updateVelocityAndDirection();
    this.resetMoveDownTimer()
}
```

Le mouvement des invaders est maintenant défini. Pourtant ceux-ci ne bougent pas encore. Pour faire bouger les invaders nous allons ajouter une fonction move dans la classe invader.

Dans la **classe invader**, ajoutez une fonction move qui prend deux paramètres (xVelocity et yVelocity)

```
move(xVelocity, yVelocity) {
    this.x += xVelocity;
    this.y += yVelocity;
}
```

Cette fonction doit être appelée depuis la fonction drawInvaders de la classe InvaderController.

```
drawInvaders(ctx){
    this.invadersRow.flat().forEach((invader)=>{
        invader.draw(ctx);
        invader.move(this.xVelocity, this.yVelocity);
    })
}
```

### Commit et push

à cette étape vos invaders bont de gauche à droite et descendent d'une ligne à chaque fin de mouvement gauche droite.

Si c'est le cas vous pouvez faire un commit avec le message "3.3.g. Mouvement des invaders"

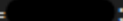

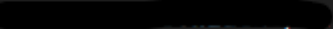





à cette étape vos invaders bont de gauche à droite et descendent d'une ligne à chaque fin de mouvement gauche droite.

Si c'est le cas vous pouvez faire un commit avec le message "3.3.g. Mouvement des invaders"

### 3.3.h. Affichage du Player

Créez un nouveau fichier `Player.js`. celui ci défini une classe `Player` qui comprend:

- un constructeur qui définit
  - une variable canvas : le canvas sur lequel dessiné le player
  - une variable velocity : vitesse du player
  - une variable x : position x de l'élément
  - une variable y: position y de l'élément
  - une variable width : largeur de l'élément
  - une variable height: hauteur de l'élément
  - une variable image : création d'un élément image associé à cette variable. La source de cette image est disponible dans le dossier images.
- une méthode draw qui permet d'afficher le player

```
constructor(canvas, velocity) {  
    this.canvas =   
    this.velocity =   
  
    this.x =   
    this.y =   
    this.width =   
    this.height =   
    this.image =   
    this.image.src =   
  
}
```

```
draw(ctx){
  ctx.drawImage(
  );
}
```

Une fois cette classe créée, **créez une nouvelle constante player dans index.js**. Cette constante sera une instantiation de la classe player avec une vitesse de 3..  
Puis dans la fonction game, faites un appel à la fonction player.draw(ctx) en lui passant le contexte du canvas en paramètre.

Vous devriez arriver à ce résultat:



### Commit et push

Si c'est le cas vous pouvez faire un commit avec le message "3.3.h.affichage du player"

### 3.3.i. Contrôle du player

Pour contrôler le player, il va falloir être à l'écoute d'événements liés aux touches du clavier.

Dans la **classe player**, ajoutez des variables (boolean) qui permettent de savoir si les touches gauche ou droite sont pressées:

- rightPressed = false;



- Ces variables seront mises à jour via une fonction appelée à la réception d'un event.

**Au niveau du constructeur**, ajoutez des listeners sur les événement keyup et keydown faisant appel à une fonction qui vérifie s'il s'agit des touches flèche gauche et droite du clavier.

```
document.addEventListener(████████████████████);
document.addEventListener(████████████████████);
```

```
keydown = (event) => {
  if (event.code == "ArrowRight") {
    this.rightPressed = true;
  }
  if (event.code == "ArrowLeft") {
    this.leftPressed = true;
  }
};
```

```
keyup = (event) => {
  if (event.code == "ArrowRight") {
    this.rightPressed = true;
  }
  if (event.code == "ArrowLeft") {
    this.leftPressed = true;
  }
};
```

La gestion des entrées clavier étant gérée, il reste à mettre à jour la position du vaisseau. Pour cela vous allez créer une fonction move dans la classe player.

```
move() {
  if (this.rightPressed) {
    this.x += this.velocity;
  } else if (this.leftPressed) {
    this.x -= this.velocity;
  }
}
```

Cette fonction est ensuite appelée depuis la fonction draw.

## Commit et push

à cette étape votre vaisseau bouge de gauche à droite lorsque vous appuyez sur les touches correspondantes.

Si c'est le cas vous pouvez faire un commit avec le message "3.3.i.contrôle du player"

Votre vaisseau est maintenant capable de bouger. En revanche, il peut sortir du canvas. Pour éviter cela, vous allez créer une nouvelle fonction qui sera appelée depuis draw. Cette fonction, `collideWithWalls`, va vérifier si la position du player dépasse les bords du canvas, et si il dépasse d'un côté ou de l'autre, il modifie la position du vaisseau.

```
collideWithWalls() {  
  //left  
  if (this.x < 0) {  
    this.x = 0;  
  }  
  
  //right  
  if (this.x > this.canvas.width - this.width) {  
    this.x = this.canvas.width - this.width;  
  }  
}
```

### Commit et push

à cette étape votre vaisseau ne sort plus de l'écran.

Si c'est le cas vous pouvez faire un commit avec le message "3.3.i.contrôle du player-mur invisible"

### 3.3.j. tir du vaisseau

Votre vaisseau doit pouvoir tirer pour anéantir les invaders.

Le tir nécessite un son.

### Comment importer un son avec webpack

il faut ajouter une règle dans le `webpack.config.json`

```
module: {  
  rules: [  
    {  
      test: /\.scss|css$/,  
      use: ['style-loader', 'css-loader', 'sass-loader'],  
    },  
    {  
      test: /\.png|svg|jpg|jpeg|gif$/i,  
      type: 'asset/resource',  
    },  
    {  
      test: /\.wav$/,  
      loader: 'file-loader',  
    },  
  ],  
}
```

puis ensuite faire l'import de le fichier qui nous intéresse:

```
import ShootSound from "../assets/sounds/shoot.wav";
```

Le chemin d'accès peut ensuite être utilisé en tant que variable ShootSound.

Peut être que cela posera problème, si oui c'est qu'il vous manque probablement un loader:

```
npm install --save-dev file-loader
```

**Dans la classe player :**

- ajoutez un boolean shootPressed initié à false.
- modifiez les fonctions keydown et keyup pour passer à true ou false la variable shootPeessed selon l'appui sur la touche "Space".

Pour gérer les tir du vaisseau vous allez avoir besoin de créer un bulletController.

**Créez un nouveau fichier BulletController.js** dans lequel vous déclarez une classe BulletController avec son constructeur.

Son constructeur va prendre plusieurs paramètres:

- canvas
- maxBulletsAtaTime
- bulletColor
- soundEnabled

le constructeur va initier plusieurs variables:

- canvas : le canvas reçu en paramètre
- maxBulletsAtaTime : le nombre maximum de bullets reçu en paramètre
- bulletColor: la couleur des tirs reçu en paramètre
- soundEnabled: le booléen reçu en paramètre
- shootSound: instancie un nouvel élément Audio avec le son disponible dans le dossier sounds.
- Le volume sonore du son est initié à 0.1.

```
constructor(canvas, maxBulletsAtaTime, bulletColor, soundEnabled) {  
  this.canvas =   
  this.maxBulletsAtaTime =   
  this   
  this   
  
  this.shootSound =   
  this.shootSound = 0.1;  
}
```

Le bulletController va garder en mémoire toutes les bullets actives.

- Pour cela vous allez créer un **tableau bullets initié à vide** dans lequel on stockera toutes les bullets instanciées.

- Pour initier un tir, vous allez ajouter une méthode shoot à la classe bulletController.

Cette fonction va prendre en paramètre:

- une position x
- une position y
- une velocity
- un temp timeTillNextBulletAllowed (initié à 0 par défaut)

Cette fonction va instancier une bullets si le nombre maximum de bullet autorisé n'est pas dépassé et si le temps entre deux bullets est inférieur ou égal à 0.

```
shoot(x, y, velocity, timeTillNextBulletAllowed = 0) {
  if (this.bullets.length < 10) {
    const bullet = new Bullet(this.canvas, x, y, velocity);
    this.bullets.push(bullet);
    if (this.soundEnabled) {
      this.shootSound.currentTime = 0;
      this.shootSound.play();
    }
    this.timeTillNextBulletAllowed = timeTillNextBulletAllowed;
  }
}
```

Dans la classe player :

- ajouter un **bulletController** en paramètre du constructeur et initiez une variable bulletController
- ajouter un **appel à la méthode shoot()** dans le cas où shootPressed est true.

```
draw(ctx) {
  this.move();
  ctx.drawImage(this.image, this.x, this.y, this.width, this.height);
  this.collideWithWalls();
  if (this.shootPressed) {
    this.bulletController.shoot(this.x + this.width / 2, this.y, 4, 10);
  }
}
```

Pour le moment la **classe Bullets** n'existe pas. Vous allez maintenant la créer:

- créer un fichier Bullet.js
- dans ce fichier une classe Bullet
- un constructeur qui prend en paramètre
  - le canvas
  - la position x
  - la position y
  - velocity
  - bulletColor

- le constructeur initialise les propriétés:
  - canvas
  - x
  - y
  - velocity
  - bulletColor
  - width (initié à 5)
  - height (initié à 20)
- une méthode draw qui prend en paramètre le contexte du canvas (ctx)

```
draw(ctx) {
  this.y -= this.velocity;
  ctx.fillStyle = this.bulletColor;
  ctx.fillRect(this.x, this.y, this.width, this.height);
}
```

Dans la Classe BulletController:

- Ajoutez un import de la classe bullet
- Afin de gérer TimeTillNextBulletAllowed, vous allez ajouter une méthode draw .

```
draw(ctx) {
  if (this.timeTillNextBulletAllowed > 0) {
    this.timeTillNextBulletAllowed--;
  }
}
```

- Toujours dans la fonction draw, pour chaque bullet instanciée vous allez y ajouter un appel à la fonction bullet.draw().

```
this.bullets.forEach((bullet) => bullet.draw(ctx));
```

Vous n'avez pas encore instancié la classe BulletController, dans index.js:

- créer une constante playerBulletController qui instancie la classe bullet controller  
`const playerBulletController = new BulletController(canvas, 10, "red", true);`
- ce playerBulletController doit être passé en paramètre de l'instanciation du Player  
`const player = new Player(canvas, 3, playerBulletController);`
- dans la fonction game, faites un appel à  
`playerBulletController.draw(ctx);`

Dans le fichier index.js faire un import de la nouvelle classe BulletController.  
 créez une nouvelle constante playerBulletController qui instancie la classe BulletController.

```
import InvaderController from "../InvaderController.js";
import Player from "../Player.js";
import BulletController from "../BulletController.js";

const canvas = document.getElementById("game");
const ctx = canvas.getContext("2d");

canvas.width = 600;
canvas.height = 600;

const background = new Image();
background.src = "images/space.png";

const playerBulletController = new BulletController(canvas, 10, "red", true);
```

Normalement l’affichage des bullets devrait maintenant fonctionner.

Le problème c’est que les bullets sortent de l’écran et ne sont jamais détruites. Il faut donc faire en sorte de les détruire si elles sortent du canvas.

Pour cela, **dans le BulletController** vous allez mettre à jour le tableau bullets avec un filtre qui se base sur la position y de la bullets pour la sortir du tableau.

```
draw(ctx) {
  this.bullets = this.bullets.filter(
    (bullet) => (bullet.y + bullet.width > 0 && bullet.y + bullet.width < this.canvas.height)
  );

  if (this.timeTillNextBulletAllowed > 0) {
    console.log(this.timeTillNextBulletAllowed);
    this.timeTillNextBulletAllowed--;
  }
  this.bullets.forEach((bullet) => bullet.draw(ctx));
}
```

### Commit et push

à cette étape votre vaisseau tire des bullets.

Si c’est le cas vous pouvez faire un commit avec le message “3.3.j.tir du vaisseau”

### 3.3.h. tir des invaders

Dans **index.js**, initiez une nouvelle constante invaderBulletController qui instancie un nouveau bulletController avec comme paramètre:

- canvas : votre canvas
- nombre maximum de bullet:4
- couleur : “white”
- soundEnabled : false

Passer ensuite invaderBulletcontroller en paramètre de l’instantiation de l’invaderController.

Dans la **classe invaderController**,

- modifiez le constructeur pour qu'il prenne en compte le bulletController
- nous allons ajouter quelques paramètres pour gérer les tirs:
  - fireBulletTimerDefault: la durée du timer entre deux tirs initié avec une valeur de 100
  - fireBulletTimer: le timer initié avec la valeur de fireBulletTimerDefault
- vous allez ajouter une méthode fireBullet
  - qui va gérer les tirs de bullet selon le timer.
  - cette fonction va être appelée depuis la fonction draw du Invader Controller.

```
fireBullet(){
  this.fireBulletTimer--;
  if(this.fireBulletTimer<=0){
    this.fireBulletTimer = this.fireBulletTimerDefault;
    const allInvaders = this.invadersRow.flat();
    const invaderIndex = Math.floor(Math.random()*allInvaders.length);
    const invader = allInvaders[invaderIndex];
    this.invaderBulletController.shoot(invader.x+invader.width/2, invader.y,-3);
  }
}
```

Pour que les bullets apparaissent, à partir du **fichier index.js**, dans la fonction game, faites un appel à invaderBulletController.draw(ctx).

### Commit et push

à cette étape votre les invaders tirent des bullets.

Si c'est le cas vous pouvez faire un commit avec le message "3.3.h.tir du vaisseau"

### 3.3.i.Collision bullets x invaders

Pour détecter les collisions, vous allez ajouter une méthode aux bullets afin qu'elle puissent renvoyer si oui ou non leur position est commune avec un autre objet.

Dans la **classe Bullets**

- créez une méthode collideWith
- cette méthode prend un paramètre "sprite"
- cette méthode renvoie un boolean si les coordonnées de la bullet coïncident avec les coordonnées du sprite.

```

collideWith(sprite) {
  if (
    this.x + this.width > sprite.x &&
    this.x < sprite.x + sprite.width &&
    this.y + this.height > sprite.y &&
    this.y < sprite.y + sprite.height
  ) {
    return true;
  } else {
    return false;
  }
}

```

cette méthode doit être appelée à chaque image depuis la classe qui contrôle l'ensemble des bullets : `bulletsController`. Pour cela, **dans la classe `bulletsController`**:

- ajoutez une méthode `collideWith`
- cette méthode prend en paramètre un sprite
- cette méthode vérifie pour chaque bullet si cette bullet est en collision avec le sprite
- si oui alors la bullet doit être retirée du tableau bullets et la fonction renvoie `true`
- si non alors la fonction renvoie `false`

```

collideWith(sprite) {
  const bulletThatHitSpriteIndex = this.bullets.findIndex((bullet) =>
    bullet.collideWith(sprite)
  );

  if (bulletThatHitSpriteIndex >= 0) {
    this.bullets.splice(bulletThatHitSpriteIndex, 1);
    return true;
  }

  return false;
}

```

Maintenant il faut que cette méthode soit appelée à chaque image avec en paramètre le sprite qui nous intéresse.

Pour cela nous allons appeler la méthode que nous venons de créer **depuis la classe `invaderController`**. Dans la classe `invaderController`,

- ajoutez le `playerBulletController` comme paramètre du constructeur
- ajoutez une variable qui récupérera ce paramètre `playerBulletController`



```

constructor(canvas, invaderBulletController, playerBulletController) {
  this.canvas = canvas;
  this.invaderBulletController = invaderBulletController;
  this.playerBulletController = playerBulletController;

  this.createInvaders();
}

```

Dans la classe invaderController:

- ajoutez une nouvelle fonction qui s'appelle collisionDetection
- cette fonction doit être appelée depuis la fonction draw de la même classe (pour être appelé à chaque frame)
- cette fonction va vérifier pour chaque invaders si il est en collision avec une bullets
- si oui, l'invaders va être retiré du tableau des invaders
- cette fonction va vérifier s'il y a des lignes d'invaders vides, auquel cas elles seront supprimées.

```

collisionDetection() {
  this.invaderRows.forEach((invaderRow) => {
    invaderRow.forEach((invader, invaderIndex) => {
      if (this.playerBulletController.isCollision(invader)) {
        invaderRow.splice(invaderIndex, 1);
      }
    });
  });

  this.invaderRows = this.invaderRows.filter((invaderRow) => invaderRow.length > 0);
}

```

### Commit et push

à cette étape vos invaders disparaissent s'ils sont touchés par des bullets..

Si c'est le cas vous pouvez faire un commit avec le message "3.3.i.collision bullet invaders"

### 3.3.j. GameOver

le game over peut apparaître dans différents cas:

- une bullets touche le player
- un invader touche le player
- tous les invaders sont éliminés

#### 3.3.j.1.une bullets touche le player

dans le fichier **index.js**:

- créez une nouvelle variable isGameOver initié à false
- dans la fonction game, conditionnez l'affichage du jeu à la valeur de isGameOver
- ajoutez une fonction checkGameOver
  - cette fonction sera appelée depuis la fonction game (gameloop) pour être appelé à chaque frame
  - la fonction vérifie si isGameOver est true, dans ce cas elle return
  - dans le cas contraire elle vérifie si une bullet est en collision avec le player.

```
let isGameOver = false;

function game() {
  checkGameOver();
  ctx.drawImage(background, 0, 0, canvas.width, canvas.height);
  displayGameOver();
  if (isGameOver) {
    invaderController.draw(ctx);
    player.draw(ctx);
    playerBulletController.draw(ctx);
    invaderBulletController.draw(ctx);
  }
}
```

```
function checkGameOver() {
  if (isGameOver) {
    return;
  }

  if (invaderBulletController.collideWith(player)) {
    isGameOver = true;
  }
}
```

votre vaisseau est maintenant sensible au bullets

### Commit et push

à cette étape votre vaisseau disparaît s'il est touché par des bullets..

Si c'est le cas vous pouvez faire un commit avec le message "3.3.j.1 collision bullet player"

### 3.3.j.2.un invader touche le player

Dans la classe **invader**:

- ajoutez une méthode collideWith qui prend en paramètre un sprite
- cette méthode vérifie si le sprite est en collision avec l'invader

```

collideWith(sprite) {
  if (
    this.x + this.width > sprite.x &&
    this.x < sprite.x + sprite.width &&
    this.y + this.height > sprite.y &&
    this.y < sprite.y + sprite.height
  ) {
    return true;
  } else {
    return false;
  }
}

```

dans **invaderController**,

- ajoutez une méthode `collideWith` qui prend en paramètre un sprite
- qui vérifie pour chaque invader si il est en collision avec un sprite

```

collideWith(sprite) {
  return this.invaderRows.flat().some((invader) => invader.collideWith(sprite));
}

```

dans **index.js**, dans la fonction `checkGameOver`:

- ajoutez une condition sur `invaderController` pour vérifier si il y a collision avec le player.

```

function checkGameOver(){
  if(isGameOver){
    return;
  }else if (invadersBulletController.collideWith(player)){
    isGameOver = true;
  }else if(invaderController.collideWith(player)){
    isGameOver = true;
  }
}

```

### Commit et push

à cette étape votre vaisseau disparaît s'il est touché par un invader.

Si c'est le cas vous pouvez faire un commit avec le message "3.3.j.2 collision invader player"

### 3.3.j.3.tous les invaders sont éliminés

dans index.js:

- créez une nouvelle variable boolean `didWin` initié à `false`
- dans la fonction `checkGameOver`, ajoutez une vérification sur le nombre de ligne d'invaders restante
- si le nombre de ligne est égale à 0 alors `didWin` passe à `true`;

```
function checkGameOver(){
  if(isGameOver){
    return;
  }else if (invadersBulletController.collideWith(player)){
    isGameOver = true;
  }else if (invaderController.collideWith(player)){
    isGameOver = true;
  }else if (invaderController.invadersRow.length===0){
    isGameOver = true;
    didWin = true;
  }
}
```

### Commit et push

À cette étape, le jeu s'arrête si tous les invaders sont neutralisés.

Si c'est le cas vous pouvez faire un commit avec le message "3.3.j.3 player win"

### 3.3.j.4.afficher le texte de fin de jeu

À la fin du jeu, deux textes peuvent être affichés à l'écran.

- le joueur perd: game over
- le joueur gagne: you win

dans **index.js**,

- créez une nouvelle méthode `displayGameOver()`
  - cette méthode est appelée depuis la `gameLoop` dans le cas ou le jeu est `gameOver`
  - elle affiche un message différents selon le type de `gameOver`

```
function displayGameOver() {
  if (isGameOver) {
    let text = didWin ? "You Win" : "Game Over";
    text+=" press enter to restart";
    let textOffset = didWin ? 3.5 : 5;

    ctx.fillStyle = "white";
    ctx.font = "70px Arial";
    ctx.fillText(text, canvas.width / textOffset, canvas.height / 2);
  }
}
```

### Commit et push

À cette étape, un texte s'affiche en fin de jeu selon que le joueur gagne ou perd.  
Si c'est le cas vous pouvez faire un commit avec le message "3.3.j.4 gameover text"

**Félicitations, vous avez terminé la V1 de votre jeu !!!**



Cette version est donc la V1.0.0. Vous allez donc:

- mettre à jour le fichier package.json avec le numéro de version qui convient
- tester votre build : `npm run build`
- créer une branche v.1.0.0 sur git pour garder cette version.

## 4.nouvelles features dans les autres versions

Maintenant que la V1 fonctionne : vous pouvez améliorer le jeu en ajoutant les fonctionnalités suivantes.

Après chaque fonctionnalité commit et push, demander une code review à l'un de vos collègues.

### Définition code review

Une code review, également appelée revue de code, est une pratique courante dans le développement logiciel où un développeur soumet son code à l'examen de ses pairs.

L'objectif principal de cette revue est d'identifier:

- les erreurs,
- les bogues,
- les inefficacités de conception
- les problèmes de sécurité dans le code

Cela permet d'améliorer la qualité du code, d'assurer sa conformité aux normes de codage de l'entreprise ou de l'équipe, et de favoriser l'apprentissage et le partage des connaissances au sein de l'équipe de développement.

Les revues de code peuvent être effectuées de manière informelle entre collègues ou de manière plus structurée avec des outils spécifiques pour faciliter le processus.

### V1.1 ajout d'un bouton replay

à la fin du jeu, que le joueur perde ou gagne, faire en sorte qu'il puisse rejouer via une interaction de votre choix. un clique sur un bouton, l'appui d'une touche...

à faire:

- intégrer la fonctionnalité (commit et push sur votre branche de version v1.1.0)
- envoyer votre code à review à un autre étudiant
- prendre en compte les retours sur une branche v1.1.1

### V1.2 ajout du score

Ajoutez un système de score à votre jeu. Le score doit apparaître à l'écran et s'incrémenter à chaque invader neutralisé.

à faire:

- intégrer la fonctionnalité (commit et push sur votre branche de version v1.2.0)
- envoyer votre code à review à un autre étudiant
- prendre en compte les retours sur une branche v1.2.1

## V1.3 ajout de nouvelle map invader

Dans votre jeu, il y a une seule “map” d’invader, c’est-à-dire un seul pattern de disposition d’invaders.

L’idée de la fonctionnalité et d’en ajouter d’autres, et que le pattern soit sélectionné au hasard en début de partie.

à faire:

- intégrer la fonctionnalité (commit et push sur votre branche de version v1.3.0)
- envoyer votre code à review à un autre étudiant
- prendre en compte les retours sur une branche v1.3.1

## V1.4 ajout de niveau de difficulté

Dans votre jeu, il y a un niveau de difficulté. L’idée ici est de proposer une nouvelle vague d’enemy qui vont plus vite une fois la première vague terminée.

Vous pouvez ainsi avoir une difficulté croissante à l’infinie jusqu’au game over du joueur.

à faire:

- intégrer la fonctionnalité (commit et push sur votre branche de version v1.4.0)
- envoyer votre code à review à un autre étudiant
- prendre en compte les retours sur une branche v1.4.1

références

<https://la-cascade.io/articles/npm-guide-complet-pour-le-debutant>  
**Tutoriel Webpack 5 pour les débutants - Gekkode**