De Révolution à Evolution
Création d'un jeu vidéo sur les thèmes de l'écologie et de la transition énergétique

Niels Lachat, 3MG01
Mentor : Patrick Rickli
Lycée Denis-de-Rougemont

2017-2018

Table des matières

1 Introduction

Le but de ce travail de maturité était de créer un jeu vidéo abordant les thématiques de l'environnement et de la transition énergétique. Le genre du jeu de gestion a été choisi pour démontrer les principes économiques de la transition énergétique. Ces principes ont bien entendu été simplifiés et modélisés afin de pouvoir les intégrer dans un jeu vidéo.

Je présenterai tout d'abord le concept sur lequel le jeu a été basé et j'expliquerai ensuite comment le joueur interagit avec le jeu. Je conclurai par une explication du fonctionnement de certaines parties du code du jeu.

2 Concept du jeu

2.1 Développement du concept

Mon projet était de créer un jeu de gestion de ressources dont l'univers du jeu était aussi proche que possible de la réalité. Le joueur devait être amené au fil des expériences du jeu à réaliser qu'à un certain moment, une transition énergétique était nécessaire afin d'assurer la survie de l'espèce humaine. J'expliquerai dans cette section comment s'est développé le concept de mon jeu.

2.1.1 La mécanique économique

La mécanique économique étant la mécanique principale de jeu, il était primordiale qu'elle soit bien pensée. Plusieurs mécaniques économiques m'ont parues intéressantes mais je n'en ai retenu qu'une seule pour les raisons décrites ci-dessous.

La première alternative consistait à avoir deux types de centrales : des centrales produisant de l'énergie [1] et d'autres produisant des ressources [2]. Le joueur aurait installé des centrales énergétiques pour produire de l'énergie. Cette énergie aurait permis de faire fonctionner ses centrales produisant des ressources. Ainsi, un cycle économique se serait créé. Deux raisons m'ont convaincu de ne pas choisir cette alternative. Premièrement, l'idée ci-dessus ne transmettait pas le message que je voulais transmettre : en effet, ce n'est pas ce que l'on produit et ce que l'on consomme qui pose un problème, c'est la façon dont nous le faisons. La deuxième raison m'ayant fait renoncer à cette alternative est la complexité des mécaniques de jeu et par conséquent du code qu'impliquait cette mécanique économique.

1. Centrales à charbon, barrages hydroélectriques, centrales nucléaires, etc...
2. Usines de textile, forges, champs, etc...

La deuxième alternative était basé sur deux types de centrales : les centrales de production de ressources énergétiques [3] et les centrales de production d'énergie (voir note de bas de page n°1). Le joueur aurait dû trouver un bon équilibre entre les ressources énergétiques qu'il aurait produites et les centrales énergétiques qui auraient consommé ces ressources. Cette alternative aurait pu être intéressante du point de vue du joueur mais elle était également trop compliquée à intégrer dans le jeu et ne reflétait pas non plus l'idée que je voulais transmettre (les centrales de production de ressources énergétiques n'apportant pas beaucoup à la réflexion).

J'ai donc opté pour la troisième alternative qui consiste à n'avoir qu'un seul type de centrales : les centrales de production d'énergie. Ainsi, le joueur choisit où il installe ses centrales et celles-ci produisent de l'énergie qui est convertie directement en mondios (nom de la monnaie du jeu). Il peut ainsi acheter de nouvelles centrales avec l'argent qu'il obtient.

2.1.2 Le personnage incarné par le joueur

Je me suis également posé la question de savoir quel personnage le joueur devrait incarner. Dans un premier temps m'est venu l'idée que le joueur pourrait être un riche patron d'une multinationale. Cependant, cela impliquait que c'était uniquement du ressort des multinationales d'assurer ou non la transition énergétique et d'effectuer des actions en faveur de l'écologie, ce qui est en tout cas partiellement faux. Cette possibilité excluait également le joueur, qui ne se sentait que très peu concerné par la problématique.

L'autre possibilité était donc que le joueur incarne le président/la présidente du monde. Cette option a l'avantage d'expliquer le pouvoir immense qu'a le joueur et de montrer que n'importe qui peut agir pour le bien de la planète. Un inconvénient de ce choix est que cela éloigne le monde du jeu de la réalité car le monde n'est évidemment pas gouverné par un président mondial. Cette alternative a tout de même été choisie pour simplifier la narration et le concept.

2.1.3 La monnaie du jeu

Une question qui s'est posée assez rapidement dans le développement de l'idée du jeu était celle du choix de la monnaie du jeu. J'ai d'abord pensé à utiliser une monnaie réelle mais la question des échelles des prix des objets du jeu m'a fait douter de ce choix. En choisissant une monnaie fictive, j'avais le choix de fixer les prix que je voulais car on ne pourrait pas comparer avec des prix réels. J'ai donc choisi de créer les *Mondios*, monnaie universelle de la planète (voir figure 1).

3. Mines, puits de pétroles, etc...

Note : L'affichage de l'argent est toujours effectué en utilisant les préfixes du Système international d'unités pour réduire l'espace que celui-ci prend sur l'écran (par exemple 150'200 sera affiché comme 150.2k).

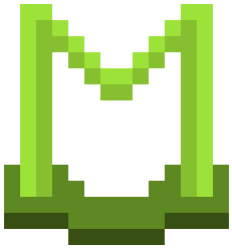

FIGURE 1 – Icône de la monnaie du jeu : les *Mondios*

2.1.4 La temporalité du jeu

J'ai assez rapidement décidé de commencer le jeu au début de la révolution industrielle (donc environ en 1800) mais j'ai eu plus de doute sur le choix de l'année de fin du jeu. La première idée qui m'est venue était de créer un jeu sans fin : le joueur aurait pu continuer de construire des centrales énergétiques jusqu'à ce qu'il n'ait plus rien à améliorer ou à construire. Toutefois, cela me semblait aller à l'encontre des principes du jeu vidéo où il y a toujours un objectif à remplir pour finir le jeu. J'ai donc choisi que le joueur gagnerait la partie s'il parvenait jusqu'à l'an 2100 sans avoir passé la limite des 2°C du réchauffement global de la planète (objectif fixé par l'Accord de Paris sur le climat, voir point [4] des références).

2.2 Concept final

Vous êtes en 1800, au début de la révolution industrielle. Vous venez d'être élu président(e) de la planète. Votre but : mener la planète à travers la révolution industrielle et assurer la transition énergétique afin de ne pas arriver à un point critique du réchauffement climatique. Le jeu s'achève soit vous avez dépassé les 2°C de réchauffement global (jeu perdu) ou si vous avez atteint l'année 2100 (jeu gagné). L'action principale du joueur consiste à gérer les centrales énergétiques présentes sur sa planète.

3 Le jeu

Dans cette section, j'expliquerai comment interagir avec le jeu en présentant les éléments de l'écran principal, les différents onglets, la vue des régions et finalement comment jouer.

3.1 Les éléments de l'écran principal

L'écran principal du jeu comporte plusieurs éléments dont je vais expliquer l'utilité ci-dessous (voir à chaque fois la figure 2).

- **L'argent du joueur** (en mondios) est affiché dans le coin en haut à gauche de l'écran.
- **L'année de jeu** est affichée en haut au centre de l'écran. Les flèches à gauche et à droite de l'année permettent de ralentir (flèche gauche) ou d'accélérer (flèche droite) le déroulement du temps du jeu.
- **Le bouton plein écran** affiché en haut à droite de l'écran permet d'activer ou de désactiver le plein écran.
- **L'onglet de recherche** est accessible via le bouton 'Recherche' situé en-dessous du bouton plein écran. Son utilité est expliquée dans la section 3.2.
- **L'onglet de statistiques** est accessible via le bouton 'Statistiques' affiché en-dessous du bouton 'Recherche'. Son utilité est également expliquée dans la section 3.2.

FIGURE 2 – Ecran principal du jeu

3.2 Les onglets du jeu

Les onglets du jeu sont affichés sous forme de journal. Il y a 3 onglets dans le jeu : l'onglet de recherche, l'onglet des statistiques et l'onglet d'achat de centrales (ce dernier sera présenté dans la section 3.3). Tous les onglets sont construits sur le même modèle, seul leur contenu change. Ainsi, chaque onglet comporte un titre (en haut au centre), un bouton permettant de fermer l'onglet (en haut à droite), trois sections (au centre), un numéro de page (en bas au centre) et des flèches pour changer de page (en bas à gauche et à droite).

L'onglet de recherche affiche les centrales énergétiques dans lesquelles le joueur peut investir des mondios (en appuyant sur le bouton vert où le prix de l'investissement est affiché) pour augmenter la probabilité de les débloquer.

FIGURE 3 – Onglet de recherche

L'onglet des statistiques affiche les différentes statistiques de jeu. Ces informations servent à guider les choix du joueur. (Voir à chaque fois les figures 4 et 5).

- La première section concerne l'argent du joueur et sa production d'argent.
- La seconde section affiche la production d'électricité des centrales du joueur.
- La troisième section affiche l'émission totale de CO_2 depuis le début du jeu et l'émission hebdomadaire de CO_2 des centrales énergétiques.
- La quatrième section (sur la deuxième page) affiche l'augmentation de température depuis le début de la révolution industrielle (début du jeu).

FIGURE 4 – Onglet des statistiques, première page

FIGURE 5 – Onglet des statistiques, deuxième page

3.3 La vue des différentes régions

En vue 'région', le joueur peut déverouiller des sites de production et interagir avec les centrales énergétiques construites (voir figure 6).

FIGURE 6 – La vue 'région'

En appuyant sur un site verouillé, une fenêtre de dialogue s'ouvre avec le prix du déverrouillage affiché (voir figure 7).

FIGURE 7 – Le déblocage d'un site de production

Il est ensuite possible de construire une centrale sur ce site de production (voir figure 8).

FIGURE 8 – L'achat d'une centrale énergétique

Dans la vue 'région', on peut améliorer une centrale énergétique pour qu'elle produise plus (ce qui aura comme conséquence qu'elle polluera également plus). Il est aussi possible de la détruire en appuyant sur le bouton rouge (voir figure 9).

FIGURE 9 – Amélioration d'une centrale énergétique

3.4 Comment jouer

Le joueur commence par devoir choisir de jouer avec ou sans tutoriel (voir figure 10). S'il choisit de suivre le tutoriel, il sera guidé dans le jeu par le personnage de Conseil qui lui expliquera le fonctionnement de tous les éléments du jeu et le but de la partie.

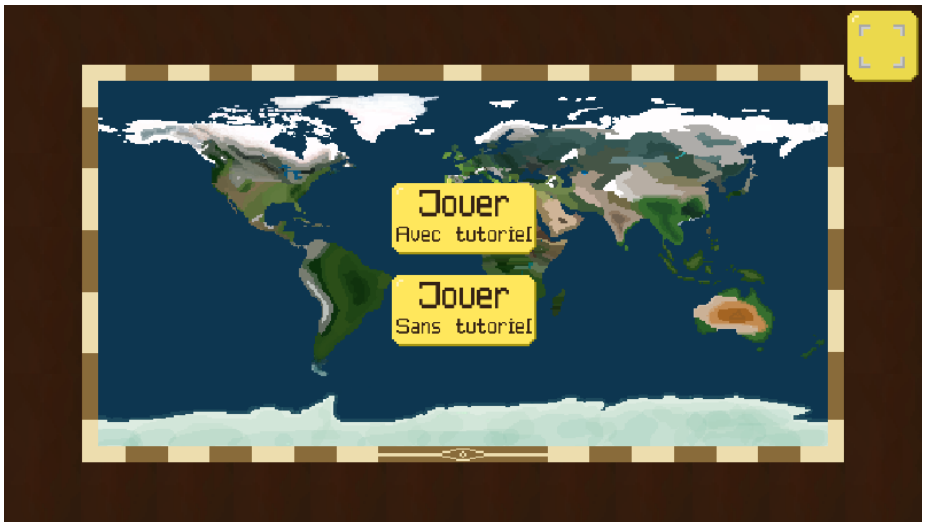

FIGURE 10 – Menu principal du jeu

Lorsqu'il commence une partie, le joueur possède 250'000 Mondios. Il devra d'abord utiliser cet argent dans l'onglet 'Recherche' (voir figure 3) qui lui permettra de débloquer une centrale énergétique. Après avoir suffisamment investi dans un type de centrale et attendu un certain moment (les centrales ne peuvent se débloquer qu'à chaque passage d'année), le type de centrale se débloque.

Après avoir débloqué un type de centrale, le joueur peut se rendre dans une région (voir figure 6) et déverrouiller un site de production (voir figure 7). Après avoir déverrouillé le site de production, le joueur peut y construire une centrale énergétique (voir figure 8). Cette centrale produira de l'énergie qui sera convertie en mondios et qui permettra d'investir dans la recherche d'autres centrales énergétiques ou de construire de nouvelles centrales. Il devra cependant faire attention à ne pas construire des centrales trop polluantes qui réchaufferaient sa planète de façon excessive (et qui lui ferait par conséquent perdre la partie).

4 Fonctionnement du code

Dans cette section, j'expliquerai quelles technologies j'ai utilisées pour créer mon jeu, comment les fichiers sont structurés et comment quelques parties du code fonctionnent. Je ne pourrai pas expliquer la totalité du code du jeu car cela serait trop long et n'apporterait pas grand chose à la compréhension du fonctionnement global du jeu.

4.1 Les technologies utilisées

Le jeu a été crée dans le langage de programmation JavaScript (avec utilisation des dernières possibilités de ECMAScript 2016, voir la référence [3]). La librairie open source Phaser JS en version 2.9.4 (voir la référence [1]) m'a servi à simplifier certaines parties du code du jeu.

Note : Les commentaires dans les scripts présentés ci-dessous ne contiennent pas de caractères avec accent pour des soucis de compatibilité des caractères.

4.2 La structure des fichiers (voir annexe A)

Les fichiers du jeu sont organisés en deux dossiers principaux : *assets* et *scripts*. Le dossier *assets* contient tous les éléments visuels du jeu (c'est-à-dire les images et les animations). Le dossier *scripts* contient tous les scripts du jeu. Les sous-dossiers de ce dossier sont *classes, data, states* et *utils*.

Classes contient toutes les définitions des classes utilisées dans le jeu.

Data contient tous les scripts qui n'ont pas vraiment une action concrète mais qui servent à stocker des données du jeu (par exemple les valeurs de puissance et de production de CO_2 des centrales).

States contient tous les fichiers dans lesquels se trouvent les *states* du jeu (voir la section 4.3 pour une explication des *states*).

Utils contient divers scripts qui sont utiles à plusieurs endroits du jeu (par exemple le fichier *globals.js*, voir la section 4.4 pour une explication).

4.3 Le script d'entrée : main.js

```
1  // Script d'entree
2
3  // configuration de Phaser.Game
4  let config = {
5      width: 800,
6      height: 450, // w/h = 16/9 pour un rapport standard
7      renderer: Phaser.CANVAS,
8      parent: "phaser-canvas",
9      antialias: false // pour que les pixel art ne soient pas
           floutes
10 };
11
12 let game = new Phaser.Game(config);
13
14 //ajout des states du jeu
15 game.state.add("boot", bootState);
16 game.state.add("preload", preloadState);
17 game.state.add("mainMenu", mainMenuState);
18 game.state.add("game", gameState);
```

```
19 game.state.add("gameEnd", gameEndState);
20
21 game.state.start("boot");
```

Ceci est le script principal car il lance l'exécution de toutes les autres parties du code. La variable *config* définit plusieurs données utiles qui seront fournies à la classe *Phaser.Game(...)* pour créer la variable *game*. Cette variable est la variable principale de la librairie Phaser : chaque fois que l'on veut rajouter un élément de jeu (image, animation, bouton, etc...), que l'on veut démarrer un état du jeu (voir prochain paragraphe) ou que l'on veut modifier des propriétés de la caméra, c'est grâce à la variable *game* qu'on le fera.

Ensuite, les différents états du jeu (*state*) sont définis. Chaque *state* sert à définir un état du jeu. Le *preloadState* représente par exemple l'étape où le jeu charge les éléments du jeu dans le cache du navigateur. Le *mainMenuState* représente l'état du jeu lorsqu'on est dans le menu principal (voir figure 10). Le premier paramètre de la méthode *game.state.add(...)* est une chaîne de caractères qui représente la clé de l'état (grâce à laquelle on pourra démarrer plus tard cet état). Le second paramètre fait référence à un objet qui fera office de conteneur pour le state.

Pour finir on appelle *game.state.start("boot")* pour démarrer le boot state (pas très important pour la compréhension du fonctionnement global du jeu).

4.4 Les propriétés globales : globals.js

```
1  //ce script contient des variables globales reutilisees tout au
       long du jeu
2
3  let globals = {
4      UI: {
5          buttonOffset: 3,
6          smallButtonScale: 2,
7
8          posBtnFontSize: 26,
9
10         shortTweenDur: 300,
11         longTweenDur: 800
12     },
13
14     //voir classes/regions.js
15     regions: {},//contient les instances de la classe Region
16     currentRegion: "", //"" quand en worldview et la key de la
           region sinon
17
18     //contient des variable en rapport avec les sites de production
19     sites: {
20         id: 0, //identifiant unique pour chaque site de production
               (voir classes/regions.js)
21         instances: []
22     },
23     factories: {
24         maxLevel: 3,
```

```
        upgradeCoeff: 1.3
    },

    data: {}, //voir scripts/data

    initMoney: 25e4,
    CO2Limit: 1e7,
    globalWarmingLimit: 2, //en degres celsius
    beginYear:1799,
    endYear: 2100,
    gameWon: undefined, //type : bool
    gameEnded: false, //pour ne pas appeler plusieurs fois gameEls.
        fadeCam(...)

    moneyMgr: null,
    productionMgr: null,
    researchMgr: null,
    ecoActionsMgr: null,

    reset: null, // defini ci-dessous,
    initData: null, //defini ci-dessous

    signals: {
        onNewspaperOpen: new Phaser.Signal(),
        onNewspaperClosed: new Phaser.Signal(),
        onFactoryUnlocked: new Phaser.Signal(),
        onRegionEntered: new Phaser.Signal(),
        onNPBtnClicked: new Phaser.Signal()
    },

    showTutorial: true
};
```

Ce script contient toutes les variables globales réutilisées durant tout le jeu. Il serait trop long de décrire l'utilité de toutes les propriétés et méthodes de l'objet *globals*. C'est pourquoi je ne décrirai que certaines d'entre elles.

- *initMoney* représente l'argent qu'a le joueur au début du jeu.
- *CO2Limit* représente la quantité maximum de CO_2 qu'il peut y avoir dans l'atmosphère. Passé cette limite, le joueur aura perdu la partie.
- *globalWarmingLimit* représente la limite de réchauffement de la planète. Le réchauffement de la planète est calculé à partir de la quantité de CO_2 dans l'atmosphère.
- *beginYear* et *endYear* représentent les années de début et de fin du jeu.
- *signals* est un objet qui contient les signaux utiles pour le tutoriel.

4.5 Le fichier principal du jeu : game.js

```
let gameState = {};

gameState.create = function(){

    gameEls.setup.background(); //voir utils/gameEls.js
```

```
6       gameEls.setup.earthMap();
7       gameEls.setup.UI();
8
9       globReg.init(); //voir utils/globReg.js
10
11      globals.moneyMgr = new MoneyMgr(globals.initMoney);
12
13      globals.productionMgr = new ProductionMgr();
14
15      globals.timeMgr = new TimeMgr(32e3, [
16          () => {
17              globals.productionMgr.update();
18
19              // ATTENTION si on change le titre du newspaper stats
20              if(gameEls.newspaper && gameEls.newspaper.data.title ==
                    "Statistiques"){
21                  globals.data.stats = updateStatsData();
22                  gameEls.newspaper.softUpdate(globals.data.stats);
23              }
24          }
25      ]);
26      globals.timeMgr.startUpdate();
27
28      globals.researchMgr = new ResearchMgr();
29
30      if(globals.showTutorial){
31          globals.tutorial = new Tutorial();
32          globals.tutorial.start();
33      }
34  };
35
36  gameState.update = function(){
37      //on ne peut que cliquer sur les regions lorsqu'il n'y a pas de
            newspaper
38      if(gameEls.newspaper == undefined){
39          globReg.update();
40      }
41  }
```

La variable principale de ce fichier se nomme *gameState*. C'est cette variable qui a été passée en argument de *game.state.add(...)* dans le fichier main.js (voir section 4.3).

4.5.1 Les *states* Phaser

Chaque *state* dans Phaser est constitué de quatre méthodes principales : *preload, create, update* et *render*.

La méthode *preload* sert à charger les éléments du jeu dans le cache du navigateur. Cependant, comme j'ai déjà chargé tous les éléments dont j'avais besoin dans ma *state* appelée *preloadState*, je n'ai plus besoin de le faire dans *gameState*.

La méthode *create* sert à exécuter des instructions qui doivent s'exécuter une seule fois avant l'appel de la méthode *update*.

La méthode *update* est appelée toutes les quelques millisecondes et permet d'effectuer des actions répétées.

La méthode *render* permet d'exécuter des méthode de débogage.

4.5.2 La méthode gameState.create

Le premier bloc de code dans cette section sert à mettre en place les différents éléments de l'interface : l'arrière-plan, la carte du monde et les autres éléments de l'interface utilisateur (UI).

Ensuite, *globReg.init()* permet d'initialiser les zones cliquables des régions de la carte et les sites de production des régions.

Les prochaines lignes initialisent les différents *managers*. Ceux-ci sont utilisés durant tout le jeu et gèrent plusieurs données importantes. Le *moneyMgr* sert entre autre à stocker l'argent total possédé par le joueur et possède une méthode qui permet de tester si on peut effectuer un achat d'une certaine valeur. *productionMgr, timeMgr* et *researchMgr* seront expliqués ci-après (voir respectivement les sections 4.6, 4.7 et 4.8).

On démarre ensuite le tutoriel si la variable *globals.showTutorial* a la valeur *true* (celle-ci a reçue la valeur vrai ou faux lors du clique sur le bouton du menu principal, voir figure 10).

4.5.3 La méthode gameState.update

Dans cette méthode, la méthode *globReg.update()* est appelée si il n'y pas d'instance de la classe *Newspaper* active (on ne veut pas que le joueur puisse cliquer à travers un onglet). *globReg.update()* permet de constamment tester si le joueur a cliqué sur une des régions et d'enclencher le zoom sur cette région si c'est le cas.

4.6 Le gestionnaire de production : productionMgr.js

```
1  class ProductionMgr{
2      constructor(){
3          this._totPower = 0;
4          this._CO2Production = 0;
5
6          this._totCO2 = 0; //CO2 emi au total dans l'atmosphere
7
8          this._powerToMondio = 1.2;
9      }
10
11     update(){
12         let totPower = 0;
13         let CO2Production = 0;
```

```
14          for(let s of globals.sites.instances){
15              totPower += s.fac.power || 0;
16              CO2Production += s.fac.CO2Production || 0;
17          }
18          this._totPower = totPower;
19          this._CO2Production = CO2Production;
20
21          globals.moneyMgr.totVal += this.mondioProduction;
22          this._totCO2 += CO2Production;
23
24          if(!globals.gameEnded){
25              if(this._totCO2 >= globals.CO2Limit){
26                  gameEls.fadeCam(2000, 1, () => {
27                      globals.gameWon = false;
28                      game.state.start("gameEnd");
29                  });
30                  globals.gameEnded = true;
31              }else{
32                  const fadeScale = 3 / 5;
33                  gameEls.fadeCam(10, fadeScale * (this._totCO2 /
                        globals.CO2Limit));
34              }
35          }
36      }
37
38      get mondioProduction(){
39          return this._totPower * this._powerToMondio;
40      }
41
42      get globalWarming() {
43          return (this._totCO2 / globals.CO2Limit) * globals.
                globalWarmingLimit;
44      }
45      //...
```

Le fichier productionMgr.js contient uniquement la classe ProductionMgr (la notation de classe telle qu'écrite ici est une nouvelle possibilité de ECMAScript 2015 [4]).

4.6.1 La méthode *constructor*

La première méthode de n'importe quelle classe est la méthode *constructor*. Celle-ci est appelée automatiquement lorsqu'on instancie un objet de cette classe. Ici, plusieurs propriétés des instances de cette classe sont initialisées (le tiret bas devant les propriétés indiquent que celles-ci ne devraient normalement pas être accédées depuis l'extérieur de l'instance de la classe). La propriété *_totPower* représente la puissance cumulée de toutes les centrales construites du jeu. *_CO2Production* représente la production de CO_2 totale des centrales. *_totCO2* représente la quantité totale de CO_2 émise depuis le début du jeu. *_powerToMondio* représente le taux de conversion de puissance à mondios (utilisé dans *get mondioProduction*, voir section 4.6.3).

4. Pour plus d'informations sur les classes, voir `https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Classes`. Site consulté le 03.03.2018.

4.6.2 La méthode *update*

Les premières lignes servent à récupérer les données des sites de production pour pouvoir mettre à jour les propriétés *_totPower* et *_CO2Production*.

On augmente ensuite la valeur totale de l'argent du joueur et le total du CO_2 relâché dans l'atmosphère.

Les prochaines lignes testent si la valeur totale de CO_2 excède le maximum de la valeur possible. Si c'est le cas, on démarre le *state "gameEnd"* qui indique que la partie est perdue. On met également à jour l'opacité de la caméra pour avoir un écran de plus en plus sombre en fonction de la valeur de *_totCO2*.

Cette méthode est appelée dans la mise à jour courte du gestionnaire de temps (voir *timeMgr*, section 4.7).

4.6.3 La méthode *get mondioProduction*

Cette méthode renvoie simplement la valeur de production de mondios (utilisée dans la méthode *update*, voir section 4.6.2). Elle utilise le mot-clé *get* pour que la méthode agisse comme si c'était une propriété de la classe.

4.6.4 La méthode *get globalWarming*

Cette méthode renvoie la valeur du réchauffement global de la planète en °C. Elle est proportionnelle à la quantité totale de CO_2 dans l'atmosphère.

4.7 Le gestionnaire du temps : timeMgr.js

```
1  class TimeMgr{
2      constructor(yearDuration, callbacks){
3          this._CONSTANTS = {
4              msYearDuration: 32000,
5              secYearDuration: 32,
6
7              maxTimeScale: 8,
8              minTimeScale: 1/4
9          };
10
11         this._yearDuration = yearDuration;
12         this._timer = game.time.create();
13
14         this._callbacks = callbacks; //array de callbacks
15         this._YUCallbacks = [];
16
17         this._year = globals.beginYear;
18         this._lastYUSec = -1;
19
20         this._timeScale = 1;
```

```
21      }
22
23      //donne le temps entre chaque loop de this._timerLoop
24      get _delay(){
25          let delay = this._yearDuration / (2*this._CONSTANTS.
                secYearDuration);
26          return delay; //donc le delay de base est de 500ms (pour un
                 yearDuration de 32'000ms)
27      }
28
29      startUpdate(){
30          this._timerLoop = this._timer.loop(this._delay, () => {
31              for(let c of this._callbacks){
32                  c();
33              }
34
35              let curSec = Math.floor(this._timer.seconds);
36
37              if(Math.floor(this._timer.seconds) % (this.
                    _yearDuration/1000) === 0 && this._lastYUSec !=
                    curSec){
38                  this._yearUpdate();
39                  this._lastYUSec = curSec;
40              }
41          }, this);
42          this._timer.start();
43
44          this._startYearDisplay();
45      }
46
47      _yearUpdate(){
48          if(this._year >= globals.endYear && !globals.gameEnded){
49              gameEls.fadeCam(2000, 1, () => {
50                  globals.gameWon = true;
51                  game.state.start("gameEnd");
52              });
53              globals.gameEnded = true;
54          }
55
56          for(let c of this._YUCallbacks){
57              c();
58          }
59
60          this._year++;
61          this._updateYearDisplay();
62      }
63      //...
```

Cette classe définit le gestionnaire de la temporalité du jeu. J'expliquerai ici brièvement les méthodes principales de cette classe. Le gestionnaire de temps gère la mise à jour courte (qui dure 500ms de base [5] et dans laquelle se produit par exemple la mise à jour de l'argent du joueur) et la mise à jour annuelle (qui lance par exemple le déblocage aléatoire des centrales énergétiques, voir section 4.8).

4.7.1 La méthode *constructor*

Dans cette méthode je commence par définir des constantes de la classe. Les deux premières constantes définissent la durée de base d'une année. Ensuite j'indique l'échelle maximale et minimale de l'accélération ou du ralentissement du temps du jeu (modifié par les boutons à côté de l'affichage de l'année, voir section 3.1).

J'assigne ensuite la valeur de la durée de l'année qui sera modifiée durant le jeu. Je crée ensuite une instance de *Phaser.Timer* que j'assigne à *this._timer*.

Je crée ensuite les arrays qui contiendront les callbacks de la mise à jour courte (500ms de base) et de la mise à jour annuelle.

Je définis ensuite la propriété *_year* qui contiendra l'année actuelle et une propriété *_lastYUSec* dont l'utilité sera expliquée plus tard.

Enfin, je crée la variable *_timeScale* qui contient l'échelle de vitesse du temps qui pourra être modifiée par le joueur.

4.7.2 La méthode *get _delay*

Cette méthode renvoie la durée d'une mise à jour courte (cette méthode sera utilisée dans la méthode *startUpdate*, voir section 4.7.3).

4.7.3 La méthode *startUpdate*

Cette méthode définit la boucle *this._timerLoop* qui s'exécutera toutes les 500ms de base. Dans cette boucle, on commence par appeler toutes les callbacks [6] présentes dans *this._callbacks*. Pour savoir s'il y a un passage d'année, on teste si le compteur (en secondes) du timer est un multiple de la durée de l'année et si on ne vient pas de faire un passage d'année (d'où l'utilité de *_lastYUSec*). Si cette condition est vraie, on appelle procède à la mise à jour annuelle (*this._yearUpdate*). On démarre ensuite le timer (de la mise à jour courte) et l'affichage de l'année.

5. Cette valeur sera modifiée si le joueur décide d'accélérer ou de ralentir le jeu.
6. Une callback est simplement une fonction exécutée après un certain événement.

4.7.4 La méthode *_yearUpdate*

Dans cette méthode, on commence par tester si l'année actuelle est plus grande (ou égale) à l'année de fin du jeu. Si c'est le cas, on fait un fondu vers un écran noir de la caméra et on démarre le *state gameEnd* (qui est l'écran de fin du jeu). Si ce n'est pas le cas, on appelle toutes les callbacks de *this._YUCallbacks*, on augmente la valeur de l'année et on met à jour l'affichage de l'année.

4.8 Le gestionnaire de recherche : researchMgr.js

```
1  class ResearchMgr{
2      constructor(){
3          //plus c est grand, plus la somme partielle augmente
               rapidement
4          this._unlockData = {
5              coalPlant: {c: 10},
6              fuelPlant: {c: 2},
7              gasPlant: {c: 2},
8              hydroPlant: {c: 5},
9              fissionPlant: {c: .2},
10             windTurbines: {c: .1},
11             solarPanels: {c: .05},
12             geothermalPlant: {c: .02},
13             fusionPlant: {c: .01}
14         };
15         this._setupUnlockData();
16
17         globals.timeMgr.addYUCallback(this.rndUnlockUpdate);
18     }
19
20     increaseUnlockProb(facType){
21         let facProb = this._unlockData[facType];
22         let {c,r,k} = facProb;
23         let ak = c * r**k;
24         facProb.partSum = Phaser.Math.roundTo(facProb.partSum + ak,
               -3);
25         facProb.k++;
26         this.getFacObj(facType).unlockProb = facProb.partSum;
27
28         // ATTENTION si on change le 'purpose' du newspaper
29         if(gameEls.newspaper && gameEls.newspaper.data.purpose == "
               factoryResearch"){
30             gameEls.newspaper.softUpdate(globals.data.
                   factoryResearch);
31         }
32     }
33
34     //methode appellee a chaque 'year update' de timeMgr
35     rndUnlockUpdate(){
36         //utilisation de 'globals' car appel depuis timeMgr
37         let unlockData = globals.researchMgr._unlockData;
38
39         //parcours des unlockProbs pour debloquer aleatoirement un
               type de centrale
```

```
40          let unlockedFacNames = []; //array de strings
41          for (let key in unlockData){
42              let facProb = unlockData[key];
43              let rnd = Phaser.Math.random(0,100);
44              if(rnd < facProb.partSum){
45                  //ajoute le nom dans la liste des centrales
                        debloquees
46                  let title = globals.researchMgr.getFacObj(key).
                        title; //obtenir le nom des centrales
47                  unlockedFacNames.push(title);
48
49                  facProb.partSum = -Infinity;
50                  globals.researchMgr.unlockFacType(key);
51              }
52          }
53          //...
```

Cette classe définit le fonctionnement de la recherche de centrales énergétiques. Chaque type de centrale a une probabilité de se débloquer (valant initialement 0) que le joueur peut augmenter en investissant dans la recherche de ce type de centrale. J'ai choisi [7] d'utiliser une suite géométrique pour représenter la valeur de la probabilité car cela présente l'avantage que la probabilité peut tendre vers 100% mais ne jamais l'atteindre complètement (ce qui laisse une part de hasard). La valeur de la probabilité est une somme partielle d'une suite géométrique.

$$a_k = c \cdot r^k \tag{1}$$

$$a_0 = c \cdot r^0 = c \tag{2}$$

$$\sum_{k=0}^{\infty} a_k = c \cdot \frac{1}{1-r} = 100, \text{ pour } |r| < 1 \tag{3}$$

$$r = 1 - \frac{c}{100} < 1 \tag{4}$$

La première équation (1) correspond au terme général de la suite géométrique. La valeur de c est égale au premier terme de la suite (2). Cela représente la valeur de la probabilité après le premier investissement et c'est donc cette variable que j'ai choisi de modifier pour obtenir différentes 'difficultés' de déblocage des centrales. Sachant que la valeur de la somme doit tendre vers 100 lorsque k tend vers l'infini (3), je peux calculer la valeur de r correspondante (4). Les formules sont tirées du *Formulaires et tables*, p.90 (voir référence [5]).

4.8.1 La méthode *constructor*

Ici je définis les valeurs de c de chaque type de centrale énergétique. Plus c est grand, plus le type de centrale sera facile à débloquer. J'initialise ensuite les autres valeurs nécessaires au calcul de la suite géométrique ($r, k = 0, partSum = 0$) grâce à la méthode *_setupUnlockData*. J'ajoute ensuite la méthode *_rndUnlockUpdate* aux mises à jour annuelles du gestionnaire du temps (cette méthode sera expliquée dans la section 4.8.3).

7. Suite à un conseil avisé de mon mentor.

4.8.2 La méthode *increaseUnlockProb*

Cette méthode sert à augmenter la probabilité de débloquer une centrale. Elle est appelée depuis le *newspaper* de recherche. Je commence par calculer le terme actuel de la série a_k que j'ajoute à la somme partielle. J'augmente ensuite la variable k. Je finis par mettre à jour l'affichage de la probabilité dans le *newspaper* de recherche (grâce à la méthode *gameEls.newspaper.softUpdate*).

4.8.3 La méthode *_rndUnlockUpdate*

Cette méthode est appelée à chaque mise à jour annuelle du gestionnaire du temps. Elle permet d'aléatoirement débloquer (ou non) une centrale énergétique. Je commence par parcourir tous les objets contenus dans *unlockData*. Je génère ensuite un nombre rationnel aléatoire entre 0 et 100. Si le nombre aléatoire est plus petit que la probabilité de déblocage de la centrale, j'ajoute le titre de la centrale dans un array des noms de centrales débloquées pour pouvoir informer la joueur de quelles types de centrales il a débloquées. J'assigne ensuite la valeur *-Infinity* à la probabilité pour que la centrale ne puisse pas être débloquée plusieurs fois. Je finis par appeler la méthode *unlockFacType* qui s'occupe de débloquer le type de centrale.

5 Conclusion

5.1 Ce que j'aurais aimé rajouter dans le jeu

Par manque de temps, il m'a été impossible d'implémenter tout ce que j'aurais voulu dans mon jeu. J'explique dans cette section quels auraient été ces ajouts.

5.1.1 Différents modes de difficulté

J'aurais voulu ajouter différents modes de difficulté pour que le jeu soit plus intéressant. J'aurais par exemple pu baisser la limite de CO_2 dans l'atmosphère avant la fin de la partie ou la probabilité de débloquer une centrale pour rendre le jeu plus difficile. J'aurais également voulu ajouter un besoin en énergie de la population qui aurait dû être satisfait, ce qui aurait rendu le jeu plus intéressant en forçant le joueur à construire plus de centrales tout en ne polluant pas trop l'atmosphère.

5.1.2 Possibilité de nouvelle partie après la fin du jeu

Une chose que j'avais prévu de faire était de pouvoir recommencer une partie après la fin du jeu sans devoir rafraîchir la page du navigateur. Cependant je me suis rendu compte que c'était plus compliqué que ce que j'imaginais (il fallait notamment réinitialiser plusieurs objets globaux).

5.1.3 Musique et effets sonores

Afin de rendre le jeu plus agréable, j'aurais voulu ajouter une musique dans le jeu et des effets sonores (lors d'un clic sur un bouton par exemple). J'aurais notamment voulu ajouter la chanson *Imagine* de John Lennon comme musique principale.

5.1.4 Réalisme et évènements historiques

Il était originalement prévu que le jeu contienne des évènement historiques qui agissent sur le déroulement de la partie mais cela aurait demandé un investissement considérable en temps que je n'avais pas. J'aurais voulu ajouter également l'explosion aléatoire des centrales nucléaires à fission pour ajouter du réalisme au jeu. Un site de production sur lequel se trouvait une centrale nucléaire explosée aurait coûté très cher à nettoyer (et aurait donc été difficilement réutilisable).

5.1.5 Actions écologiques

Une fonctionnalité majeure du jeu (que j'avais d'ailleurs commencé à développer) aurait été un onglet avec différentes actions en faveur (ou en défaveur) de l'environnement. Le joueur aurait pu accepter d'investir dans des actions en faveur de l'environnement, ce qui aurait eu un impact sur le jeu (par exemple, une action en faveur de la préservation de la forêt amazonienne aurait diminué la quantité de CO_2 dans l'atmosphère).

5.2 Ce que m'a apporté ce travail de maturité

Ce travail de maturité m'a appris plusieurs choses par rapport à la programmation. J'ai notamment appris à organiser et structurer un code d'une taille importante en différents scripts, classes et méthodes. J'ai aussi l'impression d'avoir acquis de l'expérience pour ce qui est de prévoir comment concevoir le fonctionnement d'un morceau de code pour qu'il puisse être adaptable dans le futur.

J'ai acquis également un certaine expérience dans la création de jeu vidéo (par exemple pour ce qui est de la création d'images et d'animations). Il a été également intéressant d'entendre l'avis de personnes ayant pu jouer à mon jeu. Certaines personnes m'ont fourni des critiques intéressantes grâce auxquelles j'ai parfois pu améliorer mon jeu.

5.3 Continuation du jeu

J'espère pouvoir continuer à développer mon jeu après l'avoir rendu dans cette version qui manque certaines fonctionnalités (citées dans la section 5.1). Ce travail m'a grandement plu et j'aimerais beaucoup le continuer dans le but de peut-être un jour publier mon jeu afin de contribuer à transmettre le message de l'importance de l'écologie.

Références

[1] La librairie PHASER, `http://phaser.io`, consulté le 6 septembre 2017.

[2] La documentation officielle PHASER, `https://photonstorm.github.io/phaser-ce/`, consultée régulièrement jusqu'au 18 février 2018.

[3] La version de JavaScript utilisée, `https://en.wikipedia.org/wiki/ECMAScript#7th_Edition_-_ECMAScript_2016`, consulté le 28.02.2018.

[4] Les objectifs fixés par l'*Accord de Paris sur le climat*, `https://fr.wikipedia.org/wiki/Accord_de_Paris_sur_le_climat#Objectifs`, consulté le 04.03.2018.

[5] Commission Romande de Mathématiques, *Formulaires et tables*, Éditions G d'Encre, 2015.

[6] Police de caractère utilisée dans le jeu, `https://www.searchfreefonts.com/font/advancedpixel-7.htm`, consulté le 23.09.2017.

[7] Image de base utilisée pour la carte de la terre (cette image a été modifiée), `https://minecraftworld.files.wordpress.com/2011/05/earth_flat_map.jpg`, consulté le 06.09.2017.

Annexes

A La structure des fichiers du jeu

```
.
|____assets
| |____sounds
| | |____imagine.mp3
| | |____imagine.mid
| | |____imagine.wav
| | |____imagine.ceol
| |____fonts
| | |____font.xcf
| | |____font_xml_png.zip
| | |____advancedpixel-7.ttf
| | |____pixel_font.zip
| | |____font.png
| | |____font.fnt
| |____sprites
| | |____dialogs
| | | |____dialogBox.png
| | | |____dialogBox_palette.png
| | | |____dialogBox.pyxel
```

```
| | | |____conseil.pyxel
| | |____newspaper
| | | |____newspaper.png
| | | |____newspaper.pyxel
| | |____earthMap
| | | |____earthMap.pyxel
| | | |____earthMap.png
| | | |____earthMap.jpg
| | | |____earthMap.piskel
| | |____bureau
| | | |____bureau.tmx
| | | |____tiles.pyxel
| | | |____tiles.png
| | | |____tiles2.pyxel
| | | |____bureau.png
| | |____factories
| | | |____waterAnim.png
| | | |____explosionAnim.png
| | | |____smokeAnim.png
| | | |____factories.pyxel
| | | |____sunAnim.png
| | | |____fusionAnim.png
| | | |____sunAnim.pyxel
| | | |____factories.png
| | | |____explosionAnim.pyxel
| | | |____fusionAnim.pyxel
| | | |____windAnim.png
| | | |____whiteSmokeAnim.pyxel
| | | |____waterAnim.pyxel
| | | |____smokeAnim.pyxel
| | | |____windAnim.pyxel
| | | |____whiteSmokeAnim.png
| | |____buttons
| | | |____buttons_palette.png
| | | |____pos_neg.png
| | | |____pos_neg.pyxel
| | | |____wide_buttons.pyxel
| | | |____redDot.pyxel
| | | |____close.pyxel
| | | |____earthMap_palette.png
| | | |____arrows.png
| | | |____buttons.pyxel
| | | |____buttons.png
| | | |____speed.pyxel
| | | |____speed.png
| | | |____redDot.png
```

```
| | | |____close.png
| | | |____arrows.pyxel
| | | |____wide_buttons.png
| | |____smallDialog
| | | |____box.png
| | | |____box.pyxel
| | |____mondio
| | | |____displayBox.png
| | | |____displayBox.pyxel
| | | |____icon.pyxel
| | | |____icon.png
| | |____stats
| | | |____statsIcons.pyxel
| | | |____statsIcons.png
| | |____preload
| | | |____earth.piskel
| | | |____earth.png
|____scripts
| |____classes
| | |____researchMgr.js
| | |____moneyDisplay.js
| | |____menuButton.js
| | |____newspaper.js
| | |____ecoActionsMgr.js
| | |____timeMgr.js
| | |____moneyMgr.js
| | |____dialogs.js
| | |____regions.js
| | |____factory.js
| | |____sites.js
| | |____tutorial.js
| | |____smallDialog.js
| | |____productionMgr.js
| |____data
| | |____dialogsTexts.js
| | |____stats.js
| | |____factories.js
| |____utils
| | |____gameEls.js
| | |____globReg.js
| | |____testResearchForm.js
| | |____readableNumber.js
| | |____poly.js
| | |____tmpText.js
| | |____globals.js
| | |____pos.js
```

```
| |____phaser.min.js
| |____phaser.js
| |____states
| | |____boot.js
| | |____preload.js
| | |____mainMenu.js
| | |____gameEnd.js
| | |____game.js
|____advancedpixel-7.ttf
|____index.html
|____main.js
```