Candide 2.0 ou l'optimisme en jeu vidéo

Nina Ionescu 3mg01
Mentor : Jean-Marc Ledermann
Lycée Denis de Rougemont

Table des matières

1 Introduction

Il y avait en Westphalie... Et si cet incipit mythique se retrouvait un jour pixelisé, cela donnerait quoi ? C'est ce à quoi j'ai essayé de répondre lors de ce travail de maturité, qui consistait à adapter le premier chapitre de *Candide* de Voltaire en un jeu vidéo de type RPG (jeu de rôle).

1.1 Lexique

Afin d'avoir une meilleure compréhension de ce document, voici un lexique.

assets : désigne l'ensemble des fichiers visuels et auditifs du jeu.

frame : Image constituant une partie de l'animation d'un sprite.

hp : abréviation de health points, points de vie.

map : carte.

overworld : ou monde extérieur. Désigne l'ensemble de maps constituant le jeu.

pixel art : style graphique où les assets sont dessinés à la main en respectant certains formats.

pnj : abréviation de personnage non-jouable.

sprite : image du jeu en partie transparente capable de déplacement.

spritesheet : Image regroupant les toutes les frames d'animations d'un sprite.

tile : ou carreau ; petite image carrée à texture répétée.

tilemap : ou carte à carreaux. Carte du jeu formée par des tiles.

tileset : ensemble de tiles sous forme de spritesheet.

warp : zone qui une fois pénétrée déclenche un changement de map.

2 Déroulement du jeu PAS FINI

Le joueur incarne Candide et peut se promener dans des situations inspirées du livre. Il peut dialoguer avec les pnjs présents dans les décors et faire des combats avec certains d'entre eux. Le joueur arrive dans la cour du château du Baron de Thunder-then-Tronck. sources : Phaser.js par Richard Davey Phaser-tilemap-plus par Colin Vella Bosca Ceoil par Terry Cavanagh Pyxel Edit par Danik (pseudonyme)

3 Fonctionement du code

Ce projet a été réalisé en javascript, avec les additions de ECMAScript6 et de deux librairies conçues pour la réalisation de jeu-vidéo : Phaser.js ainsi que Phaser-tilemap-plus.js . La majorité du code comprend des objets et des classes créés afin de pouvoir les utiliser comme outils de création.

3.1 Système général

Avant toute chose, il a fallu concevoir un système qui permette d'atteindre diverses variables au travers de tous les outils du jeu. C'est pourquoi l'objet "globals" (présent dans globals.js), une variable globale a été créée. Cet objet stocke toutes les variables afin d'y avoir accès facilement sans devoir se soucier des scopes des différentes classes et de leur méthodes.

(Pour retenir les diverses actions effectuées par le joueur ainsi que des données importantes telles que son positionnement ou le décor actuellement chargé à l'écran, un autre objet global (présent dans gameRef.js) est utilisé. Il sert de référence pour initialiser le personnage incarné par le joueur dans l'état où ce dernier l'avait laissé.)

(pas encore sûre, voir le local storage)

Afin d'étendre les possibilités futures de diffusion du projet, une option de traduction a été pensée dans le code de départ. Il s'agit d'un chiffre (0 pour le français, 1 pour l'anglais, ...) stocké dans l'objet mentionné au paragraphe ci-dessus. Grâce à cette variable, la fonction de dialogList.js retourne les textes figurants dans le jeu dans leur bonne version.

```
1  function setDialog(langue){
2     switch(langue){
3          case 0 :
4          globals.dialogs.myChar=["Bonjour"];
5     break;
6          case 1:
7          globals.dialogs.myChar=["Hello"];
8     break;
9   }
10 }
```

Exemple de la fonction traductrice pour le dialogue d'un personnage quelconque

3.2 Gestion des states du jeu

Les states, propres à Phaser.js, représentent une section du jeu. Chaque objet state a son lot de fonctions Phaser, à savoir preload, create, update et render. L'utilisation de ces fonctions va être décrite dans les paragraphes ci-dessous.

3.2.1 Boot et Preload

La state "Boot" fait démarrer le jeu. Elle déclenche la state "Preload". Preload est une state qui utilise uniquement la fonction preload, qui sert à associer tous les assets du jeu à un nom et les stocker dans le cache pour pouvoir les utiliser dans le code. Une fois tous les éléments stockés, la state "Menu principal" est enclenchée. Le choix d'utiliser une unique state pour ce travail permet une meilleure organisation au sein du code.

3.2.2 Menu Principal

La state "menu principal" permet au joueur de lancer le jeu ou d'en modifier la langue. Les boutons qui permettent de faire une action sont un objet Phaser. L'ensemble des boutons est initié dans une seule fonction. Cette dernière dépend de la langue que le joueur a choisi. Chaque bouton a sa propre spritesheet, ce qui pourrait poser problème au niveau de la taille et du temps si d'autres langues sont ajoutées. Une option serait de générer dans le code, les textes dans les boutons à la place de les dessiner à la main.

3.2.3 Game

La state "Game" utilise les fonctions create et update. Dans la fonction create, toutes les actions de jeu sont initiées : le terrain et le personnage sont initiés et les textes du jeu sont traduits.
La fonction update est une fonction qui est appelée chaque milliseconde. Sont présentes dans cette fonction, toutes les fonctions qui servent à interagir avec le joueur, c'est à dire la détection de la collision, les changements de maps au contact d'une zone de warp et l'apparition des bulles de dialogues des personnages non-jouables quand le joueur les approche. La fonction qui sert à faire marcher le joueur est aussi présente dans la fonction update.

3.2.4 Battle

La state "battle" est utilisée lors des combats pour initier l'interface de combat. Une nouvelle state est utilisée car il est plus facile de créer à

partir d'une state vide que de modifier une state déjà existante. En effet, la state de combat ne requiert pas une détection de collision ni la création de personnages non-jouables.
Une map de fond est initiée dans la fonction create, ainsi que les personnages à l'écran et leur barres de vie. Le combat est ensuite géré par une classe, battleManager, dont le fonctionnement va être décrit plus loin.

3.2.5 Transition avec texte

Cette state sert uniquement à afficher du texte. Une fonction est passée en argument lorsqu'on fait appel à cette state qui l'exécutera. Cette state est une alternative narrative aux cinématiques, trop difficiles à programmer.

3.3 Joueur

Le joueur est un sprite avec qui peut se déplacer lorsqu'une touche directionnelle est enfoncée. Ces déplacements sont activés uniquement quand la propriété "canMove" = true. Cette propriété sert à ce que le personnage ne puisse pas se déplacer pendant les dialogues. Le sprite possède 4 animations de marche qui sont jouées quand l'image se déplace.

FIGURE 1 – le sprite du joueur avec les différentes frames des animations de marche

Les déplacements sont détectés dans la fonction updatePlayer appellée dans la state "game".
Ce sprite possède un corps physique (spécificité de Phaser) qui permet les collisions avec son environnement.

Le sprite possède également de la vie et des attaques, qui vont être utilisées durant les combats.

3.4 Les pnjs

3.4.1 Personnages normaux

Les pnjs sont une nouvelle classe étendue du sprite. Pour créer un nouveau pnj, il faut utiliser :

```
1 var myChar = new Pnj(x,y,key,frame,name,dialogs,faceAnimKey);
```

Les arguments x et y définissent la position du sprite, la "key" est la référence de la spritesheet, frame est un numéro qui définit quelle frame sera affichée (habituellement 0 pour avoir le personnage de face),name est le nom sous forme de string du personnage, dialogs est une variable qui contient les répliques du personnages et faceAnimKey est la référence à la spritesheet correspondante aux animations pendant un dialogue.
Chaque pnj a une méthode update qui permet de détecter le joueur et créer une interaction avec lui. Le code regarde si le rectangle formé par le sprite du joueur intersecte un autre rectangle plus grand, qui englobe le pnj. Si c'est le cas et que le phylactère du pnj n'est pas encore à l'écran, le personnage crée une bulle. Lorsque la bulle est affichée, le joueur peut cliquer "enter" et faire démarrer le dialogue. Cette méthode de détection de bulle a été codée avec des "if" et beaucoup de variables booléennes. Une amélioration serait possible avec des signaux, une spécificité de Phaser, qui permettent le déclenchement d'une fonction lorsque qu'une condition est remplie.

3.4.2 Ennemis

Les ennemis sont une classe étendue des pnjs. Ce sont des pnjs dotés de points de vie et d'attaques pour pouvoir les utiliser lors de combats. Chaque ennemi possède une méthode turn, utilisée lors d'un tour au combat et une méthode startCombat qui fait basculer le jeu sur la state de combat.

3.5 Les bulles

Les bulles sont une classe. Chaque bulle est composée de trois parties, le fond, le texte et un triangle animé. Les bulles jouent plusieurs rôles. Elles peuvent indiquer un dialogue mais aussi permettre au joueur de rentrer dans

une nouvelle map (grâce à la méthode goToHouse) lorsqu'il y a un élément tel qu'une porte que le joueur doit ouvrir.

3.6 Gestion des dialogues

3.7 Gestion du terrain

3.8 Gestion des combats

3.9 Interaction avec les objets

à compléter idée : un objet possède ou non un callback

4 Scénario alternatif

expliquer les dialogues, conserver le contenu tout en l'adaptant etc...donner des exemples de situations alternatives . et le résultat actuel expliquer la complication lié au temps, le rabotage etc..

5 Conception des assets

5.1 Visuel

5.1.1 Police personnalisée

5.1.2 Sprites de dialogues

techniques(couches, dessins - restrictions), exemple

5.1.3 Sprites d'overworld

techniques, exemples

5.1.4 Tilesets

techniques, exemples

5.1.5 Tilemap

5.2 Musical

logiciel utilisé , techniques (trouver une suite d'accord, les instruments etc..) , éventuellement les bruitages , la campanella midi file

6 Conclusion