CHEVALIER Nathan

COLOR DIMENSION

Le programme étant en constante évolution, de possibles changements peuvent avoir été effectués depuis l'écriture de ce dossier. Cependant, la version présentée par ce dossier est suffisante pour l'évaluation.

Site internet : http://nathanchevalier.com/colordimension
GitHub: https://github.com/NathanCHEVALIER/Color-Dimension

Bibliothèques utilisées dans le projet :

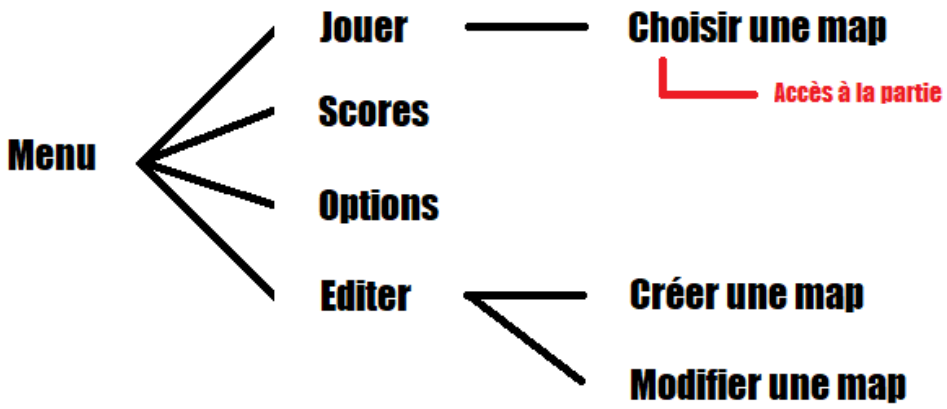
- **Pygame :** GUI Python tournée vers le jeu 2D
- **JSON** : permettant la traduction des fichiers JavaScript Object Notation (.json)

Objectifs du projet :

L'objectif de ce projet est la conception d'un jeu vidéo de plateforme, nommé Color Dimension, dans lequel Leïla, une jeune licorne, usera de ses pouvoirs, qui lui permettent de se déplacer dans la dimension des couleurs, pour passer les niveaux qui lui sont présentés… Il faudra au jeu toutes les caractéristiques d'un jeu de plateforme moderne : un jeu dynamique avec un prise en main facile, des ennemis, un éditeur de map simple d'utilisation, une sauvegarde des meilleurs scores et un rendu graphique esthétique (Haute Définition et design).

Structure du projet :

Les déplacements dans le menu et ses sous-menus se feront d'après le modèle suivant :



Répartition du travail :

Notre groupe se composent de trois personnes, Armand PICARD, Jules ECARD et moi-même. Les rôles dans ce projet sont donc répartis entre nous trois. Ces rôles sont :

- Pour Armand PICARD : les déplacements de personnage et de l'IA ennemie, les collisions avec les obstacles, la "dimension couleur" et les appels de fonctions permettant de passer du jeu au menu, et vice-versa.
- Pour Nathan CHEVALIER : la gestion de tout ce qui se réfère à l'environnement de jeu. C'est-à-dire la gestion et l'affichage de la map et l'installation des obstacles, mais aussi la création de l'éditeur de map permettant d'en créer et d'en modifier, sans oublier la sauvegarde de ces maps créées et la création des décors des maps.
- Pour Jules ECARD : la création de la page « menu », options et scores, leurs aspects graphiques, avec la sauvegarde et l'affichage des meilleurs scores dans un dossier tiers, permettant leur récupération à tous moments, même après fermeture et réouverture du jeu.

Pour réaliser ce projet de jeu vidéo, notre groupe et ses membres avons décidés de s'orienter vers de la programmation orientée objet, une structure de programmation permettant d'obtenir un code source organisé et structuré en classes, consistant en l'instanciation et l'utilisation de briques logicielles appelées objets. Ainsi nous pouvons travailler séparément sur nos parties respectives tout en rassemblant facilement ces parties dans le programme collectif.

Déroulement du projet :

Les premières séances ont été consacrés au choix d'un projet que l'on voulant intéressant, contenant un concept original, assez complet pour utiliser notre potentiel en programmation sans être impossible en termes de temps investi. Une fois décidés, il a fallu prendre en main la librairie Pygame ainsi que la syntaxe de POO en Python (Armand et moi-même avions déjà programmé en POO sous d'autres langages).

Je me suis ensuite attelé à l'affichage des maps. Ces maps se devaient d'être dynamique, c'est-à-dire qu'elles peuvent être modifiée. Pour cela, j'ai fait le choix d'enregistrer les maps sous formes de fichiers JSON et j'ai décidé d'une norme. Le choix du JSON s'explique par le fait que c'est un format que j'avais déjà utilisé, très bien supporté par Python et permettant d'enregistrer tout types de données. J'ai créé la première map du jeu manuellement afin de commencer par l'affichage de cette map dans le jeu et permettre à Armand de gérer les collisions entre notre héroïne et les obstacles.

Chaque map est subdivisionnée en différentes zone afin de limiter le chargement des décors et le calcul des collisions.

```
{"tower":  {"limit": [0, 0, 6000, 12000, "ff0000"],
            "Z0":  {"limit": [1000, 9170, 4000, 2000],
                    "plateforme":      {"0": [0, 1950, 4000, 50],
                                        "1": [2500, 1800, 400, 50],
                                        "2": [2200, 1450, 300, 50],
                                        "3": [3000, 1900, 400, 50],
                                        "4": [1300, 1700, 200, 50]
                                       },
                    "piege":           {"0": [2000, 1370, 100, 30],
                                        "1": [1800, 1920, 100, 30]
                                       },
                    "colorPlateforme": {"0": [1700, 1350, 200, 50, 800],
                                        "1": [1300, 1350, 200, 50, 300],
                                        "2": [1100, 1150, 200, 50, 900]
                                       }
                           }
               }
}
```

Première Map au format JSON : La liste des plateformes est sous la forme :

`[x, y, w, h]`

Avec x, la position en abscisse, y la position en ordonnée, w la longueur et h la hauteur

J'ai ensuite programmé la class Map qui permet d'afficher et considérer comme obstacles les différentes plateformes et pièges. C'est également dans cette classe qu'est géré le déplacement de la map voulu par Armand permettant de donner l'illusion de déplacement du personnage ainsi que le chargement des différents décors créés en fonction des différentes Map. Les décors sont des sprites qui doivent être découpés dans le programme pour créer les différentes images nécessaires.

Une fois que les maps pouvaient être chargées et interprétées, je me suis penché sur un éditeur de map. Cet éditeur se scinde en deux parties :

- Création de map : utilisation de faux champs de formulaires en récupérant les évènements des touches du clavier, choix du type de décor puis manipulation du fichier JSON pour enregistrer ces modifications

```
mot = ""
for event in pygame.event.get():
        if event.type == pygame.KEYDOWN :
         lettre = event.dict['unicode']
         if ('a' <= lettre <= 'z' or 'A' <= lettre <= 'Z'):
             mot = mot + lettre
```

Programme permettant de récupérer les mots tapés au clavier.

- Edition des maps : Après avoir choisi la zone à modifier, l'utilisateur se déplace librement sur la map en cours de création, sélectionne le type d'obstacles qu'il souhaite placer puis choisi la case de destination. Un tableau à deux dimensions (colonnes dans lignes) stocke en temps réel la valeur de chaque case, initialisée par défaut à `empty`. C'est ce tableau qui est affiché dans l'éditeur. Afin d'éviter de créer un grand nombre de Surfaces Pygame, la détection de la case cliquée est calculée à partir de la position du curseur et du déplacement relatif du fond de la map.

```
mousePos = pygame.mouse.get_pos()
y = int(((self.camPos["y"] * 100) + mousePos[1] - 1000) / 50)
x = int(((self.camPos["x"] * 100) + mousePos[0] - 1000) / 100)
if x >= 0 and x < len(self.cases[0]) and y >= 0 and y < len(self.cases):
        self.cases[y][x] = [self.current["element"][0], self.current["element"][1]]
```

Extrait du programme déterminant la case modifiée

En parallèle, j'ai créé à l'aide du logiciel Inkscape (Open Source) les décors du jeu ainsi que l'animation de la licorne en SVG, un format permettant d'être facilement retravaillé.

Enfin, à la suite de l'engouement de certains de nos camarades pour notre jeu, nous avons décidés de l'exporter en exécutable (cf. Armand) et de le proposer en téléchargement. J'ai donc créé une petite page web très simpliste puisque j'ai commencé la programmation par le web.

Problèmes rencontrés :

De manière générale, je n'ai pas rencontré de gros problèmes algorithmiques au cours de l'élaboration de ce projet, cependant je me suis confronté à différent problèmes mineurs :

- Python étant un langage interprété, et du fait de l'utilisation d'une librairie lourde, nous avons dû faire de nombreuses concessions pour diminuer la puissance de calcul nécessaire, en subdivisionnant la map par exemple.
- Oublis récurrents à cause des connaissances antérieures d'autres syntaxes algorithmiques : Python ne nécessite de `;` en fin d'instruction, `elseif` devient `elif`, on doit utiliser le mot clef self pour appeler un attribut ou une méthode du même objet…
- `["empty"] * 10` crée une liste de 10 `empty`, `[["empty"] * 10] * 10` créera bien une liste de 10 listes de 10 `empty` mais cette dernière sera dupliquée, telle que `list[2][5] = "value"` équivaut à `list[*][5] = "value"`

Sources d'améliorations et ajouts futurs :

Color Dimension est le premier projet que nous réalisons à nous trois et nous souhaitons continuer son développement en dehors du cadre de l'ISN, cela étant rendu possible par le choix particulièrement judicieux que nous avons fait.

L'améliorations la plus complexe mais la plus importante est sans nul doute l'optimisation du jeu, tant au niveau du programme que de l'affichage « scintillant » afin d'obtenir un meilleur résultat en consommant moins de mémoire (processeur, RAM). Des options ainsi que de nouveaux obstacles pourront également être ajoutés au fur et à mesure du développement du jeu.

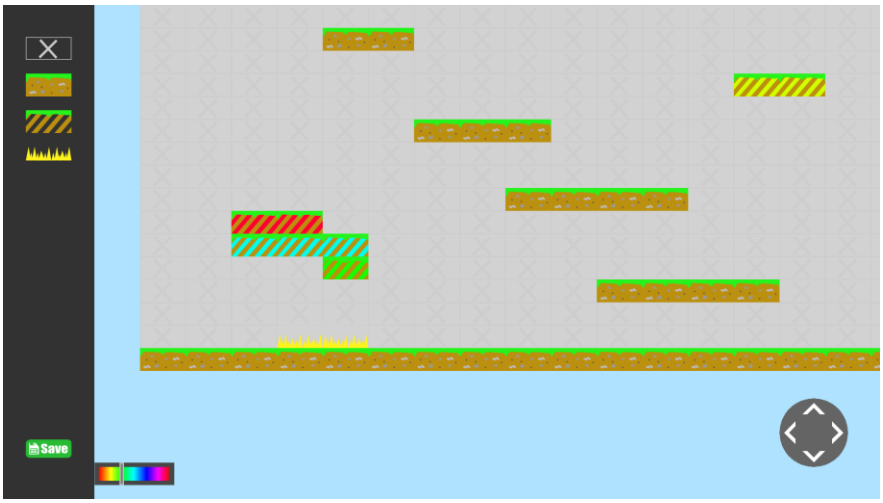
L'éditeur est à ce jour partiellement incomplet et devra être en partie revu.

Nous souhaiterions également ajouter un aspect communautaire au jeu en enregistrant les meilleurs scores des joueurs en ligne, en ajoutant la possibilité d'échanger les maps créées ou en intégrant un mode multijoueur.

Enfin, nous rencontrons régulièrement des problèmes avec les calculs physiques de collision, notre héroïne pouvant traverser certaines plateformes sous certains angles et avec une certaine vitesse.

Rendu du projet :

Menu principal du jeu



Editeur de map : L'utilisateur se déplace et pose les obstacles ou il le souhaite

Bibliographie :

- Documentation Pygame
- Stackoverflow, la référence des programmeurs si des questions subsistent
- Serveur Discord d'entraide
- Crédits musique libres de droits :
    - Andrew Applepie - Almost Winter
    - Desert Voices - Tobu
    - Happy Life - FREDJI
    - Second Nature - Audionautix
    - Sneaky Snitch - Kevin MacLeod (No Copyright Music)
    - Dance of the Sugar Plum Fairy - Kevin MacLeod
    - The Phantom's Castle by Darren Curtis

Annexe :

Sont joints avec ce document les deux fichiers de codes sources commentés sur lesquelles j'ai essentiellement travaillés, le code complet étant disponible sur l'adresse du repo de GitHub donné en introduction.