



UNIVERSITÉ
CAEN
NORMANDIE

Sokoban

Projet Conception Logicielle

Guillaume LEMONNIER / Ronan CARRE

Aboubacar KEITA / Alexis SUARD

L1 info, 2020-2021

Groupe 4A

25 mars 2021

Le jeu Sokoban est un jeu de déplacement de caisse vers un point donné. Le but est d'amener toutes les caisses sur des cibles afin de terminer le niveau.

Table des matières

1	Initialisation du Projet	3
1.1	Première approche	3
2	Vision du Projet	4
2.1	Les fonctions codés	4
2.2	Répartition du travail	4
3	Developpement du code	5
3.1	Description des structures de données	5
3.1.1	Matrice	5
3.1.2	Chose Level	5
3.1.3	Les Run	6
3.1.4	Visuel	6
3.2	Description de l'algorithme A*	6
4	Construction du Projet	7
4.1	Schéma du projet	7
4.2	Chaîne de traitement	8
5	Mise en fonctionnement	9
5.1	Côté utilisateur	9
5.1.1	Démarrage	9
5.1.2	Phase de Jeu	9
5.2	Côté A*	10
6	Conclusion	11
6.1	Résumé de la conception	11
6.2	Pour aller plus loin	11

1 Initialisation du Projet

1.1 Première approche

En première approche nous avons envisagé de créer une map avec tableau de valeur en python. Dans ce tableau il y aurait alors eu le joueur représenté par une valeur en caractère ou numérale. Chaque déplacement de l'utilisateur aurait alors changer la valeur de sa position par une valeur de vide et changer la valeur de la case ou il souhaitais aller. Il s'agissait alors d'une bonne approche pour le côté humain néanmoins elle était veine et inutile pour l'IA.

2 Vision du Projet

2.1 Les fonctions codés

Le jeu se divise en trois phases clés. La première phase se déroule lors du lancement de l'exécution du programme. Cette phase demande à l'utilisateur de sélectionner le niveau désiré à l'aide des flèches directionnelles. La deuxième phase s'occupe de l'exécution du jeu de son début jusqu'à une victoire. Enfin la dernière phase est celle qui s'occupe de demander à l'utilisateur s'il souhaite relancer un niveau ou s'il souhaite quitter le jeu.

En parallèle à l'exécution du joueur une IA s'exécute sur le coté droit, afin de mettre en compétition l'IA et le joueur.

2.2 Répartition du travail

Dans notre groupe de quatre nous nous sommes répartie le travail de manière suivante :

Guillaume s'est occupé de la partie jeu du code, car ce dernier, sur son temps personnel, avais codé des petits jeux avec pygame lui procurant alors une plus grande aisance dans l'utilisation de cette librairie. De plus Guillaume a participé à l'écriture du LaTeX.

Ronan, quand à lui, s'est occupé de coder l'IA avec l'algorithme A* car il se sentais inspiré pour codé l'IA. De plus il a écrit le Beamer pour la présentation orale et à aussi crée des maps pour le jeu en XSB.

Aboubacar s'est occupé d'une partie de l'algorithme A* avec Ronan et aussi d'une partie de la rédaction du LaTeX avec Guillaume.

Alexis à créé une parie des maps du jeu en XSB.

3 Développement du code

3.1 Description des structures de données

Le développement du jeu Sokoban comprend de nombreuses structures de données. En effet celui-ci comporte six structures de données sans compter celles dans le code de l'IA. Chaque page de code comprend une structure de donnée hormis le fichier lancement.py qui lui s'occupe uniquement de lancer l'exécution, servir de centralisation des données et de décider quel partie du programme sera joué.

3.1.1 Matrice

La première structure de donnée à être utilisé est aussi la plus importante du jeu. Il s'agit de la structure de la Matrice.py. Cette dernière est composé de tous les éléments nécessaire afin de placer le joueur sur la matrice. La class Matrice contient aussi la fonction move qui est la fonction la plus importante du jeu, en effet elle permet de tester si le déplacement est possible et si le jeu doit déplacer une caisse ou non.

```
fonction move:
    copie <- classe
    test_direction <- position + direction_voulue
    si le joueur ne va pas dans le mur:
        si le joueur pousse une caisse:
            si le joueur ne pousse pas la caisse dans le mur:
                si le joueur ne pousse pas la caisse dans une caisse:
                    recherche de la caisse qui va etre pousser : {...}
                    la pousser
                sinon:
                    ne pas avancer et retourner la copie
            sinon:
                ne pas avancer et retourner la copie
        deplacer la position du joueur
        placer les caisse sur la map vierge
        placer le joueur sur la map vierge
        deplacement + 1
        retourner la copie
    sinon:
        ne pas avancer et retourner la copie
```

Ainsi montrer la fonction move sert à la fois à faire le teste de possibilité et son déplacement si possible. La classe Matrice comptient aussi la fonction victory testant si le joueur a réussi à bien placer toutes les caisses. De ce fait la focntion teste si la coordonnée de chaque caisse correspond à la coordonnée d'une balise (étant donné que les caisse ne peuvent pas se mettre l'une sur l'autre elles auront toutes une coordonnée différente). Si la coordonnée d'une caisse correspond à la coordonnée d'une balise alors le nombre de caisse bien placé augmente de un. Enfin si la valeur de la variable des caisses bien placé correspond à la longueur de la liste des caisse alors c'est une victoire.

3.1.2 Chose Level

La deuxième classe la plus importante est la classe Chose level de chose level.py. Cette classe s'occupe de en premier lieu de tester la valeur du niveau choisie dans sa fonction selection. Si le niveau choisie est 0 alors le niveau choisie sera aléatoire sinon le niveau choisie sera le valeur entrée.

La deuxième fonction est la plus importante, en effet la fonction update ouvre d'abord le fichier XSB lié à la valeur du niveau choisi. Puis une fois cela fait la fonction prend chaque ligne du XSB et l'ajoute à une liste qui viens d'être créé et chaque valeur de chaque ligne est un élément de la liste des lignes. Ensuite la fonction teste la liste pour trouver ou se situe le caractère 'p' afin de trouver la valeur en y du personnage et ensuite elle cherche le deuxième 'p' afin de créer la coordonnée en x (programme uniquement possible en Python en effet l'avantage du python est qu'il est possible de mélanger les char et int afin de transformer le char '1' en int (1)). Ensuite le programme fais la même chose pour les caisse à un détaille pret que pour les caisse il ne s'agit pas du caractère 'p' qui est cherché mais le caractère 'c'.

3.1.3 Les Run

La troisième partie du code la plus importante est la partie des run. Ces paties étaient à la base dans lancement.py mais se sont retrouvé dans des partie différentes afin de facilité la lecture et la compréhension de lancement.py.

La première est la classe Run Level du fichier run level.py. Cette classe sert à tester la touche sur laquelle l'utilisateur appuye afin de déterminé si l'utilisateur veux selectionner le niveau supérieur ou alors le niveau inférieur. De plus cette fonction sert à envoyer la commande de création du niveau dans chose level.py. Enfin cette fonction affiche le niveau qui est actuellement pointé par l'utilisateur.

La deuxième est la classe Run de run.py. Cette classe sert à tester la fleche sur laquelle l'utilisateur appuye pendant l'état de jeu et en fonction de la touche appuyer cette classe renvoie soit None (absence de touche) ou une liste de deux valeur (-1, 0 ou 1) en fonction de la touche appuyer. Cette valeur est ensuite envoyé dans move de Matrix.py pour tenté de déplacé le personnage et/ou une caisse.

La dernière classe est la classe Win de run victory.py. Cette classe se lance uniquement si l'utilisateur gagne. Elle sert donc à demander si l'utilisateur souhaite relancer un niveau ou s'il souhaite arrêter de jouer. Si l'utilisateur choisi d'arrêter de joué alors la boucle while s'arrête et la jeu se ferme avant de terminé le programme. Et si l'utilisateur choisi de continué de joué le programme créer une nouvelle Matrice afin de recommencer une partie de zéro.

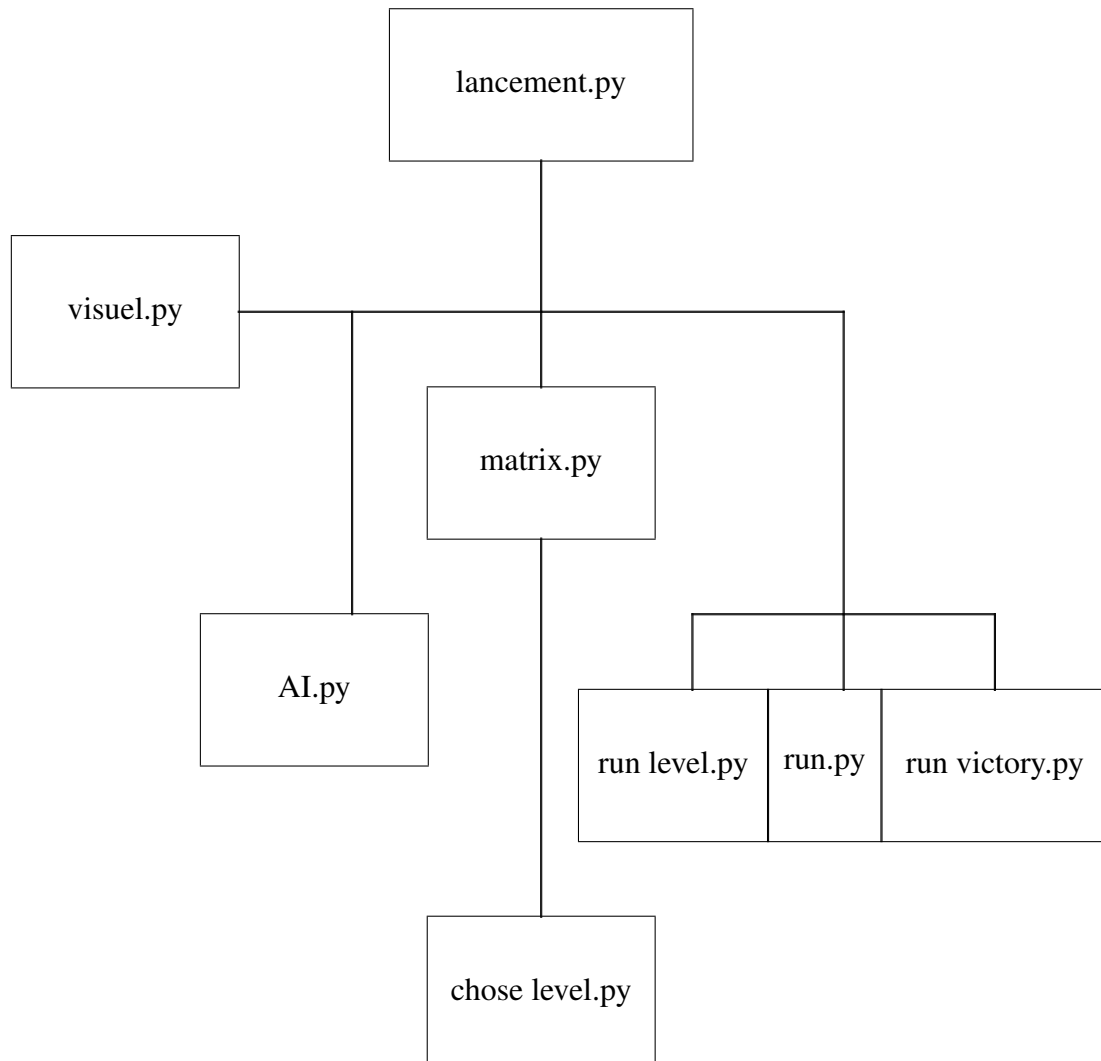
3.1.4 Visuel

La dernière classe est la classe Visuel de visuel.py. Cette classe se sert de la Matrice qui lui est arrivé en entrée pour en sortir l'élément grille afin d'avoir accès à la liste pour en construire une représentation dans le screen. Pour chaque élément de la liste il y a un élément visuel relié. Ainsi en fonction du caractère entrée le code choisira tel ou tel tuile pour le représenté.

3.2 Description de l'algorithme A*

4 Construction du Projet

4.1 Schéma du projet



4.2 Chaîne de traitement

Le projet commence en exécutant la commande `python3 lancement.py`. Ce fichier contient la fonction `while` qui s'occupe de faire tourner le jeu continuellement tant que l'utilisateur n'a pas demandé de fermer le jeu. Ce fichier sert de hub (point de ralliement) pour toutes les fonctions. Ainsi le `lancement.py` est le fichier principal mais n'est pas celui contenant les fonctions les plus importantes. `lancement.py` vérifie les états de jeu de `matrix.py` afin de savoir dans quelle situation de jeu le joueur se trouve. Si l'état de `Matrice.level running` est à `true`, alors le code exécute la classe `run level` afin de demander à l'utilisateur de sélectionner un niveau. Si l'état de `Matrice.victoire` est à `true` alors le programme lance `run victory` afin de demander à l'utilisateur s'il veut rejouer. Enfin si les deux tests sont à `false` alors la Matrice s'initialise lors du premier tour de boucle puis attend une action du joueur.

Le code de `lancement.py` entre donc dans `matrix.py` pour y effectuer des tests d'états afin de choisir quel run exécuter. Mais le code de `matrix.py` ne sert pas qu'à vérifier des états il sert aussi à exécuter le code de `chose level.py` afin de lancer un niveau à créer et de récupérer ses données.

Le code de `chose level.py` sert à ouvrir le fichier XSB sélectionné et à le transformer en plusieurs listes. Le fichier `matrix.py` a alors juste à se servir sur les listes créées par le fichier `chose level.py`.

Enfin la liste de `matrix.py` sort à chaque action pour aller dans le `lancement.py` puis dans `visuel.py` afin d'y être analysé pour en sortir un affichage pour l'utilisateur.

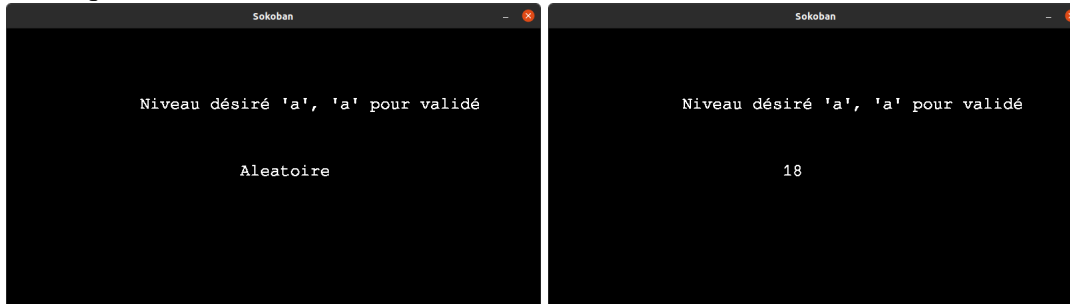
5 Mise en fonctionnement

La mise en fonctionnement est en trois phases comme le code. Les phases sont indépendantes et ont un style graphics commun avec la police de l'écriture et part leurs positionnement.

5.1 Côté utilisateur

5.1.1 Démarrage

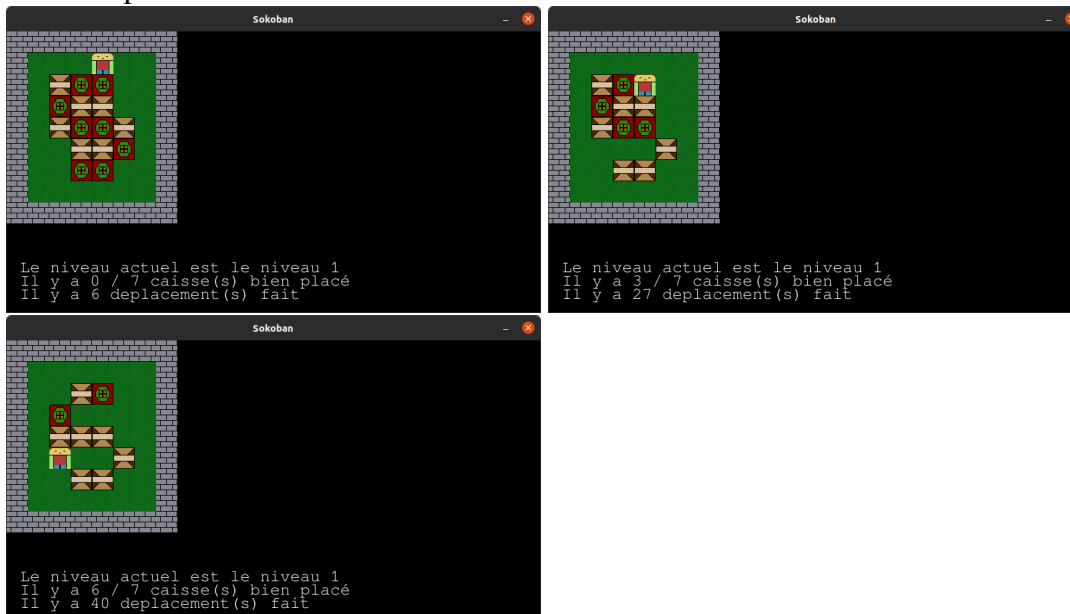
Le jeu commence par la première phase. Cette phase est la phase de selection du niveau. Il est simplement représenté par une phrase demandant quel niveau l'utilisateur souhaite exécuter ainsi que le niveau qui est actuellement ciblé.



L'utilisateur a alors uniquement la possibilité de naviguer à gauche ou à droite avec les fleches afin de selectionner le niveau. Une fois qu'il à décidé du niveau voulu il n'a plus qu'à faire un double 'a'.

5.1.2 Phase de Jeu

La phase de jeu succede la phase de choix du niveau. Lors de cette phase le joueur à la possibilité de se mouvoir sur deux axes et donc dans quatre directions. Il a aussi la possibilité de pousser des caisses mais pas de les tirer.



Les caisses sont représenté par le dessin suivant



Le personnage est représenté par le dessin suivant



Les murs sont représenté par le dessin suivant



Les cible sont représenté par le dessin suivant



Enfin les zones vides sont représenté par de l'herbe



Cette représentation est liée à la grille qui est envoyé dans la fonction update de visuel.py. Chaque représentation visuel est indépendante et redessine intégralement la fenêtre lors d'une action.

5.2 Côté A*

6 Conclusion

6.1 Résumé de la conception

La conception du jeu à été entièrement réalisé par Guillaume LEMONNIER. Bien qu'au départ il y eu une répartition du travail de manière homogène, l'absentéisme due au confinement, à changer les rôles et à changer la répartition du travail. Quand à l'IA, elle a été entièrement réalisé par Ronan CARRE pour mêmes raisons qui ont poussé Guillaume LEMONNIER à réalisé seul le jeu.

Le jeu se déroule en trois phases. La phase de selection de niveau. La phase d'exécution du niveau. Et la phase de relancement ou non du jeu.

L'IA quand à elle

6.2 Pour aller plus loin

Pour aller pour loin nous avons penssé à plusieurs améliorations tels que ajouter du sound design. Afin de rendre le jeu plus vivant et réaliste pour l'utilisateur. Nous avons aussi pensé à ajouter des animation de déplacement pour l'utilisateur, afin de rendre le jeu moins rigide visuellement mais toujours utilisable pour l'IA. Enfin nous avons pensé à créer une nouvelle phase du jeu, en parrallèle de l'execution de niveeau, un mode de création de niveau à la souris avec les blocs sur le côté. Le joueur aurait alors uniquement à selectionner une case pour en faire un élément. Suite à sa conception le niveau serait enregistré en XSB et tester par l'IA pour vérifié sa faisabilité.