UNIVERSITÉ
CAEN
NORMANDIE

Sokoban
Projet Conception Logicielle

Guillaume LEMONNIER / Ronan CARRE
Aboubacar KEITA / Alexis SUARD
L1 info, 2020-2021
Groupe 4A

13 avril 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 terminé le niveau.

Table des matières

1 Initialisation du Projet

1.1 Première approche

En première approche nous avions 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 charactère ou numéralle. Chaque déplacement de l'utilisateur aurais alors changer la valeur de sa postion 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 néanmois elle était veine et innutillle 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 de transmettre des idée pour le PDF et à aussi participé à la création des niveau.

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

3 Developpement 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'execution, 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 Matrix.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:
                    recherhce de la caisse qui va etre pousser : {...}
                        la pousser
                sinon:
                    ne pas avancer et retouner la copie
            sinon:
                ne pas avancer et retouner 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 retouner 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 touver ou se situe le charactère 'p' afin de touver 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 charactère 'p' qui est cherché mais le charactère '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 arréter de jouer. Si l'utilisateur choisi d'arréter de joué alors la boucle while s'arrète 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 recommancer 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 charactère entrée le code choisira tel ou tel tuile pour le représenté.

3.2 Description de l'algorithme A*

L'algorithme A* est un algorithme de base de l'inteligence artificielle. Cette algorithme, est un algorithme de recherche de chemain. Il est souvent utilisé dabs ke domaine du jeu video pour son éfficacité et son exactitude.

Dans notre programme l'agorithme A* se retoruve dans le fichier AIAll.py. Ce fichier contient plusieurs class indispensable pour l'execution de l'ia. La seule fonction doit etre éxécuté à l'extérieur du fichier est la focntion update de la class ExeAI. Cette fonction est le file conducteur de l'ia et lui donne tout ce qu'elle doit faire. Cette focntion est assez longue car elle est codé de manière à ce qu'elle soit très visuel par un utilisateur depuis le terminale.

Son éxécution commence par initialiser les paramètres dont elle aura besoin lors de son éxécutrion. Puis elle teste les mouvements qu'elle peut réalisé depuis son point de départ. Puis jusqu'à ce qu'elle trouve une solution parfaite elle exécute les testes en boucle.

3.3 Non trivial

3.3.1 Makefile

Makefile est type de fichier servant à réaliser des commandes terminale plus rapide. Makefile s'utilise sans extention et uniquement avec un fichier nommé Makefile.

```
SRC = lancement.py\
EXE3 = python3
EXE2 = python2
EXE = python
exe3:
	@($(EXE3) $(SRC))
exe2:
	@($(EXE2) $(SRC))
exe:
	@($(EXE) $(SRC))
```

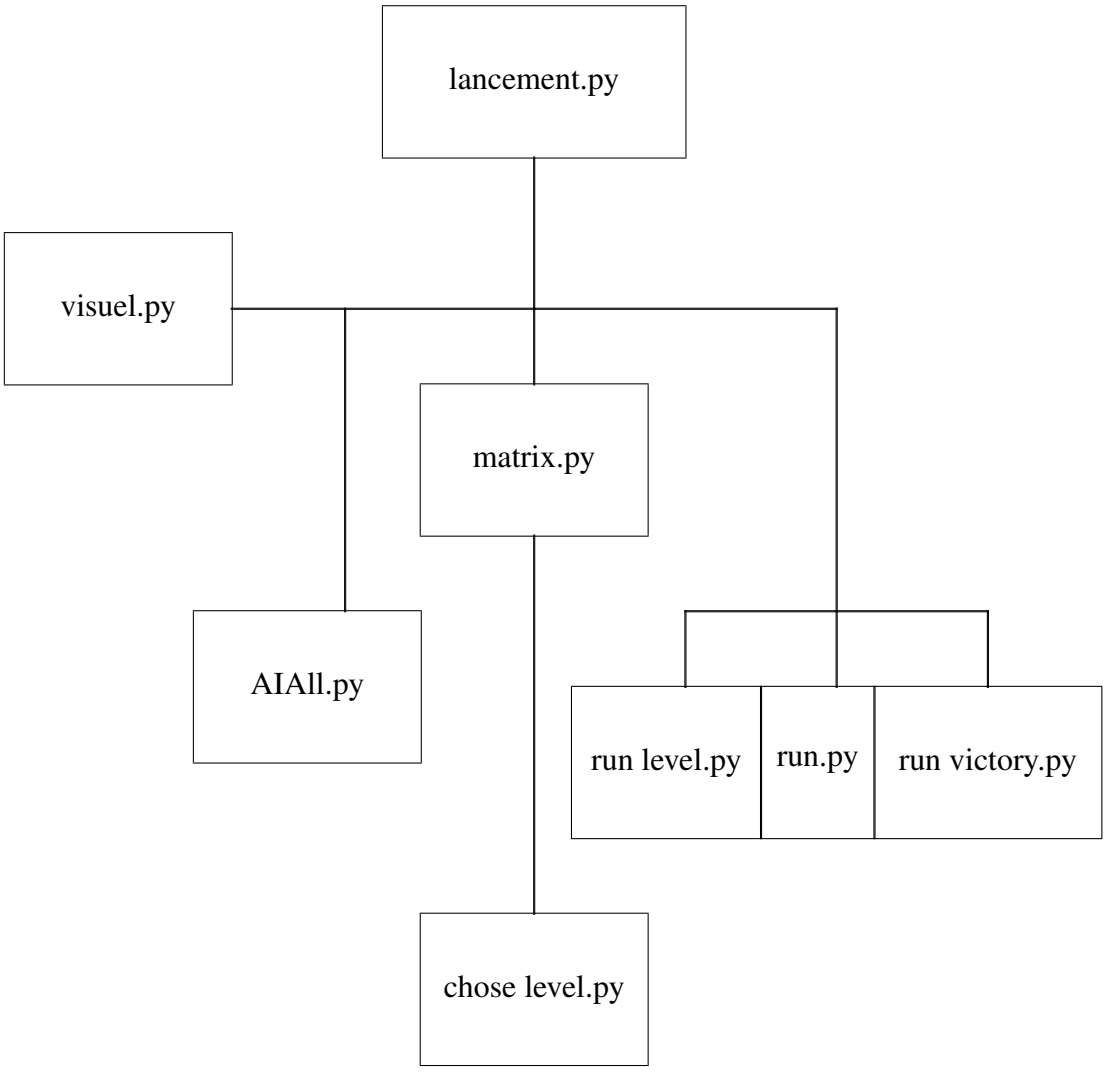
Dans ce fichier le fichier à utilisé est défini par SRC (choix personnel de le nommé comme cela mais aussi due aux normes de nommage). Il y a deux EXE afin de s'adapter aux différentes versions de python. Pour éxécuter le fichier Makefile dans le terminale il suffit d'écrire "make" suivie du nom de la fonction à utiliser (par exemple pour ma part sur python3 j'écris "make exe3" et le message qui est envoyé dans le terminale est "python3 lancement.py"). Les fichier Makefile sont souvent utilisés pour les grands projets et donc lorsqu'il y a de nombreux fichiers à executer et donc à compiler.

3.3.2 Heapq

La librairie Heapq de python est une librairie servant à trier les état dans une file de donnée. Nous l'utilisons pour l'IA, qui due à ces nombreuses possibilité de directions, afin qu'elle puissse controlé automatiquement ses arbres d'éxécutions.

4 Construction du Projet

4.1 Schéma du projet



4.2 Chaîne de traitement

Le projet commence en éxécutant 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 ralliment) pour toutes les fonction. Ainsi le lancement.py est le fichier principale 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 quel situation de jeu le joueur se trouve. Si l'état de Matrice.level running est à true, alors le code execution la class run level afin de demandé à l'utilisateur de selectionné un niveau. Si l'état de Matrice.victoire est à true alors le programme lance run victory afin de demander à l'utiisateur s'il veut rejouer. Enfin si les deux test 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 effectué des tests d'états afin de choisir quel run exécuter. Mais le code de matrix.py ne sert pas qu'à vérifié des états il sert aussi à executer 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 selectionné et à le transformer en plusieur listes. Le fichier matrix.py à alors juste à se servir sur les listes créer 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.

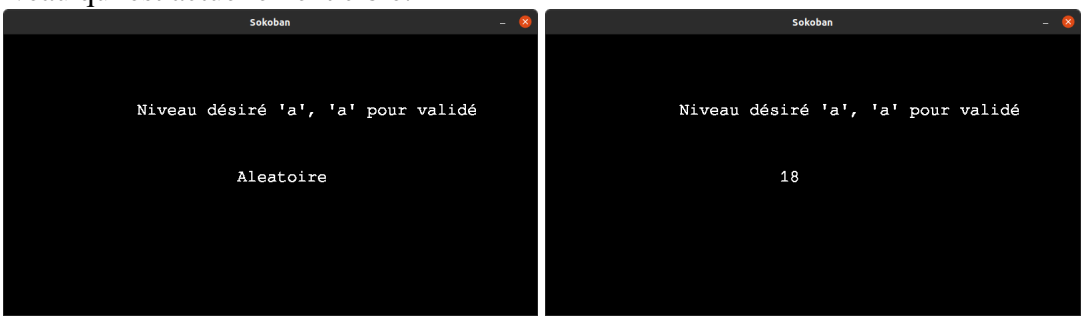
5 Mise en fonctionnement

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

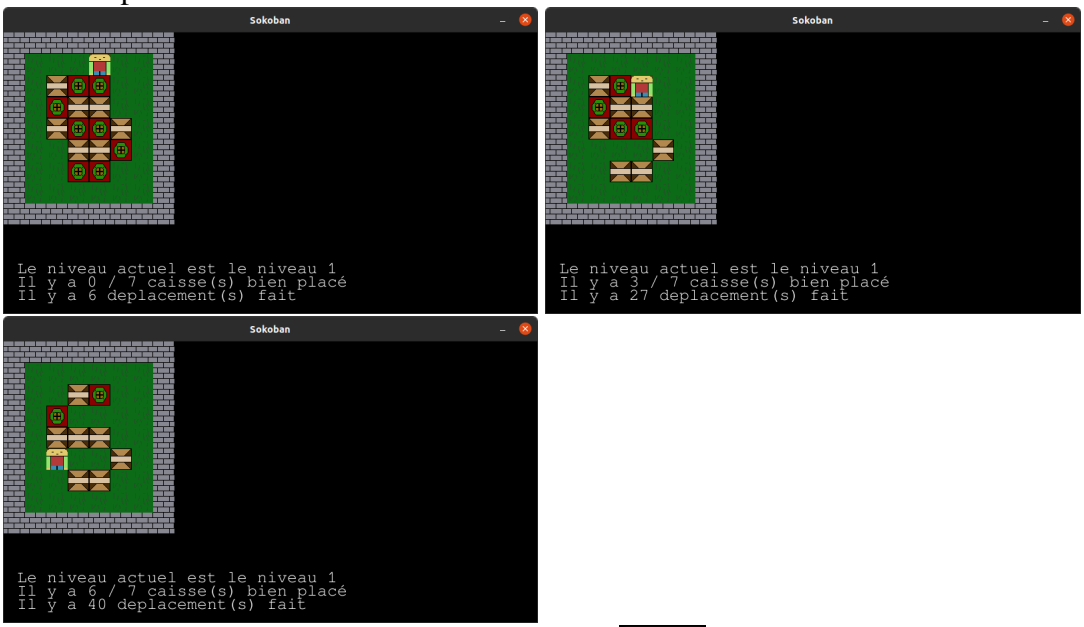
Le jeu commence par la première phase. Cette phase est la phase de selection du niveau. Il est simplement représenté par une phrase demandant quel niveau l'utilisateur souhaite éxécuter ainsi que le niveau qui est actuellement ciblé.



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 représenation visuel est indépendante et redessine intégrallement la fenêtre lors d'une action.

Lors de son fonctionnement, à chaque mouvement du joueur la matrice se met à jour et est redessiné. Alors au coeur du programme, la fonction move de la classe Matrice, crée une nouvelle Matrice qui sera alors une nouvelle Matrice.grille. Cette nouvelle Matrice.grille est une nouvelle liste de liste composé de chaque nouvelle position de chaque éléments. Cette nouvelle class est alors la class utilisé pour l'affichage de la matrice. La nouvelle Matrice se retrouve alors dans le hub du programme afin d'être analysé puis affiché dans le fichier du visuel.

5.2 Côté A*

5.2.1 Les temps de l'IA

Afin de vérifié l'éfficité de l'IA sur le programme nous avons testé tous les niveau avec l'IA s'éxécutant.

Niveau 1 : Indéfini - Indéfini
Niveau 2 : Indéfini - Indéfini
Niveau 3 : Indéfini - Indéfini
Niveau 4 : Indéfini - Indéfini
Niveau 5 : Indéfini - Indéfini
Niveau 6 : Indéfini - Indéfini
Niveau 7 : Indéfini - Indéfini
Niveau 8 : Indéfini - Indéfini
Niveau 9 : Indéfini - Indéfini
Niveau 10 : Indéfini - Indéfini
Niveau 11 : Indéfini - Indéfini
Niveau 12 : Indéfini - Indéfini
Niveau 13 : Indéfini - Indéfini
Niveau 14 : Indéfini - Indéfini
Niveau 15 : Indéfini - Indéfini
Niveau 16 : Indéfini - Indéfini
Niveau 17 : Indéfini - Indéfini
Niveau 18 : moins 1 secondes - 16 secondes
Niveau 19 : 17 minutes 16 secondes - Indéfini
Niveau 20 : 3 minutes 27 secondes - Indéfini
Niveau 21 : Indéfini - Indéfini

Il existe deux versions du code de AIAll, une version avec du texte dans le terminale afin de voir ce qui se passe, et une version sans texte afin d'améliorer sa vitesse d'éxécution. Les temps indiqué comme étant indéfini sont soit des niveau où l'ia s'arrête avant d'avoir trouvé le chemain, soit que nous n'avons pas trouvé le temps de le tester (certain teste pouvants dépasser les 10h d'éxécutions). Tous ces temps ont étaient réalisé sur un ordinateur portable équipé d'un Processeur IntelCore I5 9th 3.6GHz (cela montre ainsi que l'éxécution de l'IA nécéssite de de nombreuses étapes d'éxécutions).

5.2.2 Sortie de L'IA

La sortie de l'IA est une liste de lettre : ['e', 'e', 'e', 'e', 'e', 's', 's', 'w', 'W', 'e', 's', 's', 'W', 'e', 'e', 's', 's', 'w', 'w', 'w', 'w', 'N', 'w', 'n', 'n', 'E', 'N', 's', 'S', 'w']. Les lettres sont des références aux directions d'une rose des vents.

S et s = South

N et n = North

E et e = East

W et w = West

La différence entre les lettres majuscule et les minuscules est que les majuscule désigne le fait qu'il s'agit d'un mouvement lié à un déplacement d'une caisse.

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'absentéhisme 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 le jeu.

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

L'IA quand à elle a la possibilité d'être activé lors de la deuxième phase de jeu uniquement pour résoudre le niveau. Cependant elle ne garde pas en mémoire le niveau qu'elle à réalisé. En effet si il y a une nouvelle éxécution de l'ia sur le même niveau et cela sans redémarrer le programme, l'IA devra de nouveau faire le calcul pour le niveau.

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 avions 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 parralèle de l'execution de niveau, un mode de création de niveau à la sourie avec les blocs sur le côté. Le joueur aurais 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.