Année universitaire 2017-2018
Université de Caen Basse-Normandie

Rapport projet TPA :

L2 Informatique

Introduction

• Durant l'année 2017-2018 à l'Université de Caen, il était demandé pour les étudiants en licence 2 d'informatique de réaliser une application selon une liste donnée Pour la matière nommée TPA. Nous avons donc choisi de travailler sur le sokoban[1]. Le sokoban est un jeu vidéo où nous devons réaliser un puzzles, c'est a dire mettre la totaliser des caisses sur des zone prédéfinie. Afin d'effectuer ce travaille, il nous était indiquer de le programmer a l'aide du langage JAVA 10 et d'utiliser la Forge UniCaen[2] avec soit Git soit SVN.

1. source `https://fr.wikipedia.org/wiki/Sokoban`
2. `https://forge.info.unicaen.fr/projects/sokiban-12`

Table des matières

I Le Projet : Sokoban 1
A) Le Sokoban 1
B) La Conception du jeu 1
II Le Projet 4
A) La Réalisation du Projet 4
B) Architecture du Projet 5
C) Élément Technique 10
III Suite Au Projet 14
A) Problème obtenue 14
B) Amélioration possible 15
IV Conclusion 16

I Le Projet : Sokoban

A) Le Sokoban

• Le Sokoban est un jeu d'énigmes Labyrinthe conçu par Hiroyuki Imabayashi dans les années 80. [3]

Le but du jeu est assez simple à comprendre : L'utilisateur dirige un personnage dans un labyrinthe. Le joueur doit donc ranger des caisses sur des cases cibles. Il peut se déplacer dans les quatre directions, et pousser (mais pas tirer) une seule caisse à la fois. Une fois toutes les caisses rangées, le niveau est réussi et le joueur passe alors au niveau suivant, plus difficile en général. L'idéal est de réussir avec le moins de coups possibles (déplacements et poussées). [4]

B) La Conception du jeu

• Afin de produire le jeu dans un temps précis nous avons donc dû nous répartir les taches a effectuer. L'aspect difficile du projet est la production d'une Intelligence Artificielle(IA) A l'aide l'algorithme Astar(A*) [5] (pour plus d'information CF rapport Astar) Et la réalisation d'une interface graphique sur un langage nouveau vue cette année (Java) . Le principe du jeu reste assez simple, en effet, le personnage se déplace sur un plateau qui se comporte comme une grille à deux dimensions et doit pousser des caisses vers une direction afin de résoudre le problème. Pour cela nous avons partagé la production, en effet Arthur vérifiait si le programme fonctionnait (sur la première version), il a indiqué les erreurs afin de les corriger, quant à Nathan, il s'est occupé de la production de la première version (sur console) de plus il a géré la conception du jeu, c'est à dire le choix des différente class et leur noms. Par la suite Arthur a travailler sur l'interface graphique sous les directives et les schémas de Nathan, et quant a lui il s'occupait de l'algorithme Astar, par la suite nous nous sommes concentrés sur une seule chose afin de rendre le programme au mieux à temps.

Déjà existant

• Le jeu étant vu et revu dans de multiples application existant que ce soit dans le jeu vidéo avec des énigmes afin d'implémenter un niveau avec un gameplays différent. Il y a donc sur Internet la possibilité de récupérer des lignes de code tout fait tellement le jeu est connu dans le domaine vidéo ludique. De plus le " Sokoban Solver " est un sujet d'étude souvent pris par les étudiants en Master ou autre, car le sujet est très intéressant, car grâce à ce jeu et ça complexité, il permet d'étudier les Algorithme de recherche dans un arbre ou un graphe et étudier les graphes en eu même.

3. Pour plus d'information cf Cahier des Charges
4. Source Wikipédia `https://fr.wikipedia.org/wiki/Sokoban`
5. Source Wikipédia `hhttps://en.wikipedia.org/wiki/A*_search_algorithm`

Le Prototypage

Version prévue :

- V0 : Jeux sur inviter de Commande.
  - V0.1 : Jeux sur inviter de commande
  - V0.2 : grille+ placement personnage
  - V0.3 : ajout déplacement personnage
  - V0.4 : ajout déplacement caisse
  - V0.5 : ajout gestion collision avec mur
  - V0.6 : ajout condition victoire
  - V0.7 : ajout déplacement + collision sur plusieurs caisses

- V1 : Jeux sur Interface Graphique.
  - V1.0 : Début interface graphique
  - V1.1 : Ajout des Sprites
  - V1.2 : Ajout Interface Graphique
  - V1.3 : Ajout des déplacements du personnage avec des inputs
  - V1.4 : Ajout des déplacements avec flèche directionnelle et touche z,q,s,d
  - V1.5 : Ajout de différents boutons et HUD
  - V1.6 : Ajout de l'Interface Graphique

- V2 : Ajout de l'Intelligence Artificiel.
  - V2.0 : Recherche documentaire sur L'IA
  - V2.1 : Réalisation de L'IA
    - V2.1.1 : Essaye algorithme A*
    - V2.1.2 : Essaye algorithme de Dijkstra [a]
    - V2.1.3 : Essaye algorithme JPS [b]
  - V2.2 : Essaye réalisation du niveau par L'IA
    - V2.2.0 : Résolution niveau AStar (A*)
    - V2.2.1 : Résolution niveau Dijkstra
    - V2.2.2 : Résolution niveau JPS

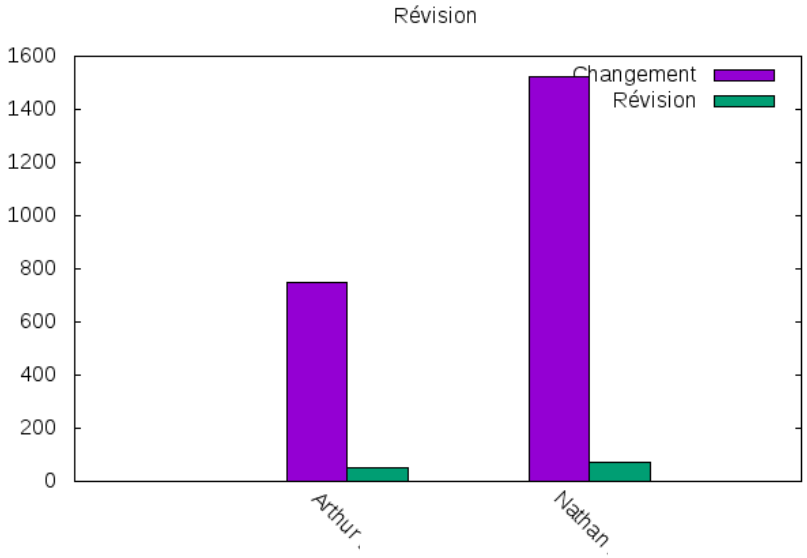
- V3 : Ajout de l'affichage et du mode versus Utilisateur vs IA
  - V3.0 : Ajout du mode versus (humain machine)
  - V3.1 : Essaye de différent mode D'affichage
  - V3.2 : fin du projet principale

- V4 : Ajout bonus

a. Source `https://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra`
b. Source `https://en.wikipedia.org/wiki/Jump_point_search`

• Afin d'avancer, Nathan a tenue tout au long du projet une fiche de version a suivre, afin d'avancer au mieux durant le projet, vous pouvez voir au-dessus les 4 versions prévues. Il y a la version 0 qui correspond à la version sur console (réaliser en duo), puis la version 1 qui correspond à l'interface Graphique(réaliser par Arthur), la version 2 qui est inintelligence Artificielle, la version 3 qui correspond à la mise en conflit l'ordinateur et l'utilisateur et pour finir la version 4 qui n'est qu'une version bonus, avec des ajouts secondaires

Voici le graphique des modifications apporter par notre groupe/



II Le Projet

A) La Réalisation du Projet

• La réalisation du projet Sokoban c'est étendu sur environ 4 mois, la réalisation du projet était libre, c'est à dire que nous étions libre de la réalisation du projet et que nous devions réaliser l'application seuls de A à Z et le rendre à la date limite qui était le 13 Avril. Afin de réaliser le projet le mot d'ordre de sa production était la régularité en effet, nous étions souvent en contact afin de voir l'avancée du projet tout cela dans le but de rendre un projet le plus parfait possible (insérer graphique statistique)

• Une fois sur le projet afin de pouvoir essayer l'application et d'obtenir le jeu, il suffisait de récupérer le niveau dans le fichier levels avec la Class Reader, une fois le plateau récupérer il suffit de récupérer la place du joueur, des caisses, des zones et de les appliquer a des Class. Pour effectuer l'application, la création du plateau, la récupération des objet ou encore les déplacements non pas été les plus dure a réaliser, en effet le plus dur que nous avons fais dans ce projet était l'utilisation des objets, en effet des erreurs nous étaient apparues car les attributs de certains objets ne bougeaient pas alors des méthodes ont été ajoutées, ces obstacles a la création du projet non pas été très dur a corriger mais le plus compliqué sur le projet restera la création de l'algorithme Astar et l'interface Graphique, afin de régler tout cela. Nous nous sommes répartis les taches, Arthur s'est occupé de l'interface et Nathan s'est occupé de l'algorithme. Pour cela Nathan a effectué des schémas afin de montrer ce qu'il attendait comme interface, c'est à dire à quoi doit ressembler l'accueil et à quoi doit ressembler l'HUD, quant a lui, il a effectué des recherches sur différents Algorithmes tels que l'algorithme de Djisktra, A*, JPS (CF fichier Algo.txt sur dossier Ressources/Idée). Une fois les recherches effectuées, la création de l'algorithme ne fut pas compliqué mais fut inutile car l'algorithme codée trouvait certes le chemin le plus optimisé mais ne permettait pas de résoudre le labyrinthe . Pour cela nous avons regardé sur internet des programmes qui résolvent automatiquement un niveau, nous nous sommes donc inspirés d'un programme trouvé sur internet afin de résoudre le programme .

• De plus toutes les images présentes sur l'application sont des dessins effectués à la main puis repris sur ordinateur, tout cela dans le but d'éviter tous les problèmes de copyright et de mette une touche d'originalité sur notre projet afin que notre projet change de tous les autres projets de Sokoban fait par les autres groupes de la promotion.

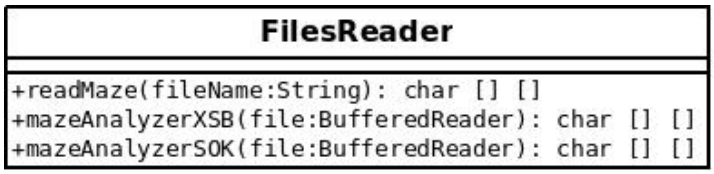
• Une fois tous cela effectué, nous avons essayé d'utiliser deux fenêtres afin de combattre le joueur et l'ordinateur et essayé de résoudre le problème avec d'autres algorithmes de pathfinding .

B) Architecture du Projet

Les Packages

La Package TextReader

• Le package TextReader correspond a toute les Class et méthodes qui consiste a l'utilisation d'un fichier, en effet sur notre projet ce package ne comporte qu'une lecture de fichier, cette Classe nommée FilesReadder possède des méthode qui nous permet de récupérer les plateau (ici nommée Board) des niveaux situer dans un fichier externe. Il est aussi possible de rajouter une Classe avec des méthode afin d'écrire dans un fichier ou autre.



La Package Levels

• Le package levels correspond a toute les Class et fichier (SOK [6] ou XSB [7]) correspondant au niveau du Sokoban, dans ce package ce trouve deux Class et un fichier (list) qui correspond a tout les niveau avec l'extension XSB, TXT ou SOK. Les deux Class qui sont contenue dans ce package sont Levels qui correspond a la création des niveaux, en effet lorsque que l'on appelle le constructeur de cette classe, on lui passe un nom et en suite utilise la méthode mazeAnalyzer de la Class fileReadder afin de récupérer le niveau correspondant au nom du niveau.

• Il y a aussi la Class Levels-list, cette Class correspond a la liste de tout les niveaux du jeux, en effet cette Class consiste a charger la totalité des niveaux présent dans le fichier "list" et les ajoute dans une ArrayList afin de les récupérer plus tard.

La Package interfaceGraphique

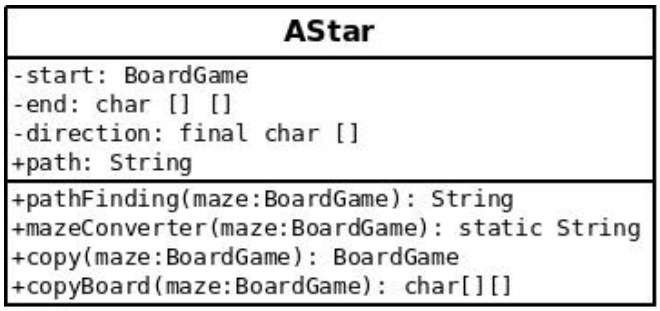
• Ce package correspond comme son nom l'indique a l'interface du jeu, vous pourrez voir les différentes Class qui correspond aux possibles écrans aux qu'elles vous pourrez voir, tels que l'accueil, les Crédit, les options, le jeu, De plus il ce trouve une Class nommée Load qui consiste a charger les différentes images présente dans le fichier 'images' et les intégrer dans le jeu ou l'interface.

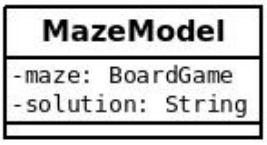
6. le format .sok pour les niveaux du sokoban, bien qu'ayant été conçu pour remplacer le format .xsb, reste peu courant

7. le format .xsb pour les niveaux du sokoban, est de loin, le plus courant

La Package AI

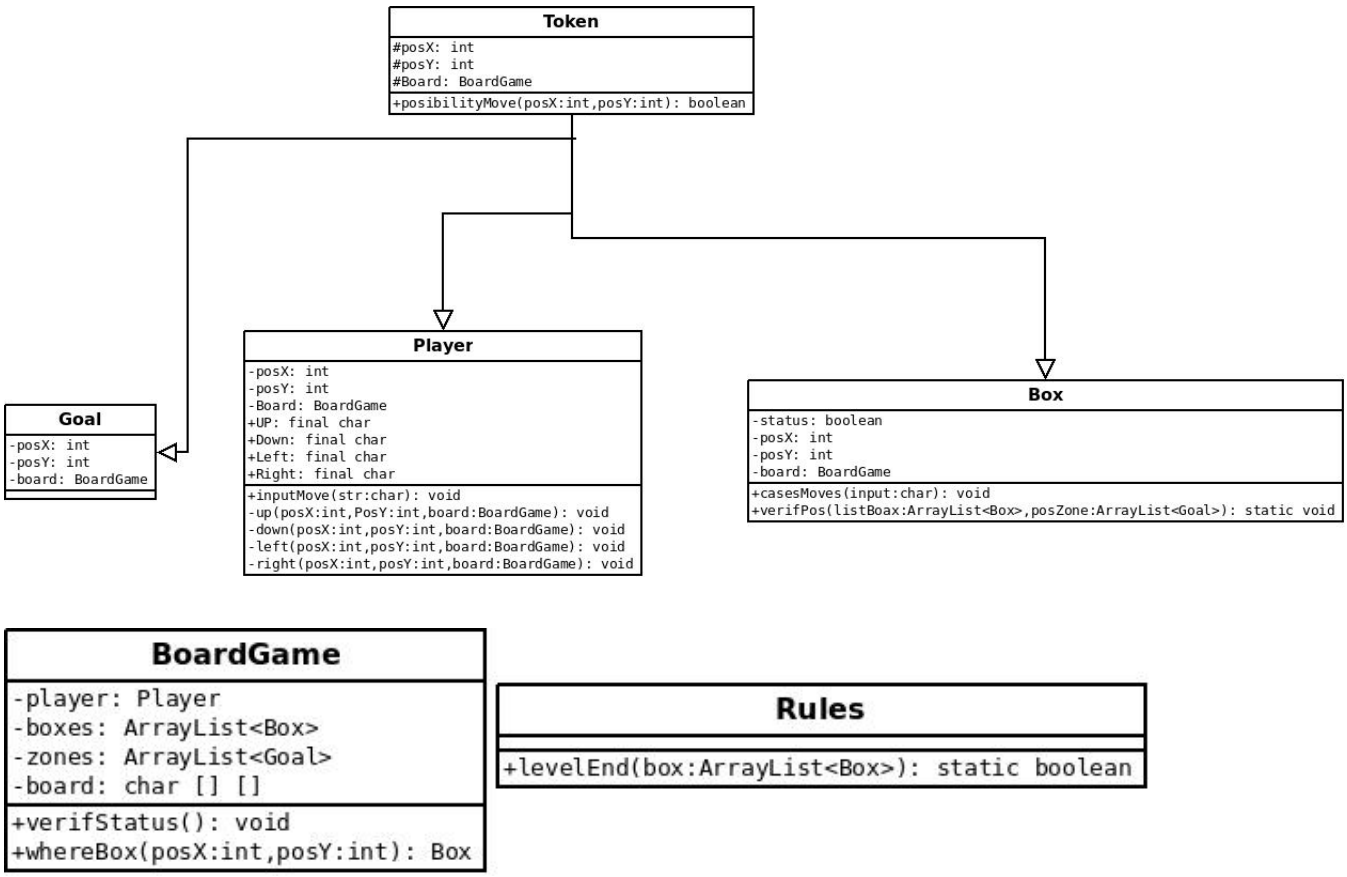
• Ce package correspond a toute les Class et méthode utile pour la réalisation de Astar sur notre projet, il y a deux Class, Astar qui correspond au méthode de pathfinding donc l'IA qui vas trouver le chemin le plus optimiser pour résoudre le niveau et MazeModel qui sert a garder en mémoire un plateau et le chemin / mouvement effectuer pour arriver a ce point





La Package Sokoban

• Le package Sokoban correspond au jeu, c'est a dire, elle contient toute les Class qui serve a la réalisation du jeu, que ce soit les règles, les pions ou le plateau. Le package est lui même séparer en trois, en effet vous trouverez dans ce package la Class exécutable / main (Sokoban.java) et deux autre package/ fichier qui sont Game et Token. Le Package Game consiste en toutes les Class qui concerne le jeu directement, c'est a dire le plateau (élément central du jeu) et les règles, quant a Token, il correspond a toutes les class correspondant a des pions sur le plateau comme le joueur, les zone et les boites, il y a même une class Token dont toutes les autre classe de ce package hérite.



Les Fonctionnalité

- Afin de réaliser le Projet, nous avons donc recherché les moyens de produire le projet avec les moyens les plus clairs et le plus pratique possible.

Class Token

- La classe Token correspond a tous les jetons du plateau, c'est a dire tous les objets présents sur le plateau sauf les sols et les murs, soit le joueur, les boites et les zones, il possède comme attribut la position du Token en X et Y et de plateau, Token possédera donc les méthodes qui vérifient les déplacements du joueur et des boites afin d'empêcher les boites de sortir du plateau ou d'écraser les murs et boites, de plus il contiendra les Getter et les Setter pour tous les jetons du plateau.

Class Player

- Pour le projet, nous avons donc créé les classes Player, qui va correspondre au personnage que le joueur contrôle, il possède comme attribut sa position en X et Y et le plateau qui le contient, il aura donc toutes les méthodes qui correspondent au déplacement, il héritera donc de la classe Token.

Class Box

- Le plateau contient aussi les objets de la Classe Box qui correspond donc aux boites, elle héritera de la Classe Token, elle possède donc les mêmes attributs et les mêmes méthodes, mais

cette classe possède un statut c'est à dire si la caisse ce situe sur une zone, si ou alors son statut devient vrai. De plus elle possède les méthodes cases Moves qui permet de modifier la position d'une caisses selon son déplacement et la méthode verifPos qui permet de vérifier si les caisses sont situées sur des zones afin de vérifier si le jeux est fini ou pas .

Class Goal

- Il y a aussi sur le plateau les objets de type Goal qui correspond aux zones du plateau.

Class BoardGame

- Le plateau est aussi un objet de type BoardGame qui possède en attribut les objets présents sur le plateau et un plateau de jeux. La class a comme méthode les méthodes qui permettent de vérifier les statuts de toutes les boites et les méthodes qui place et trouve les objets sur le plateau et biensur les setter et getter.

Class Rules

- Il y a aussi la Class Rules qui géré les règles du jeux tels que le tour des joueurs ou si le jeu est fini.

Class AStar et MazeModel

- De plus, il y a toutes les classes qui permettent la résolution du jeu a l'aide de l'algorithme A*, comme la Class Astar qui contient la méthode de pathfinding, donc Astar (l'algorithme) et les méthodes qui permet de copier tout le BoardGame actuel comme une méthode qui permet de copier le board de type char [][] et de recréer un nouveau objet de type BoardGame et une méthode qui transforme le board de type char [] [] en un String. De plus, il y a la méthode qui permet la comparaison entre deux objets de type char [] []. Il y a aussi la class mazeModel qui ne sert qu'a garder en mémoire le boardGame actuel et le chemin/ mouvement utiliser afin d'arriver à cet état depuis l'état initial.

Class Levels et Levels-list

- Dans l'application il y aussi les class correspondante aux niveaux, elles sont contenues dans dans le package levels, elle correspond au class Levels et Level-list, chacune de ces deux class servent a charger un niveau (class Levels) a l'aide de son nom et de son constructeur, quant a l'autre, elle sert a charger tout les niveaux présent dans un fichier, c'est a dire créer les objet Levels avec un nom type et les mettre dans une liste afin de récupérer le niveau pour en charger un nouveau.

Class TextReader

- Les deux Class précédentes se servent de la Class TextReadder du package du même nom. En effet, cette Class sert à lire dans un fichier, ici les fichiers sont dans le fichier list dans le package Levels, afin de récupérer sont contenue et de l'intégrer a un objet de type Levels. Dans cette Class, il y a deux types de lecture, en effet sur le Sokoban, il existe trois type de fichier propre a ce jeu, les fichiers XSB, SOK et STB. Dans le projet, il y a la possibilité de lire les

fichier de type XSB et SOK a l'aide des méthode mazeAnalyzerXSB et mazeAnalyserSOK, car chacun des fichiers est différent l'un de l'autre. SOK a pour première ligne la longueur et la largeur du niveau quant a XSB il commence par ; et le nom du niveau.

- Et bien sûr il y a dans presque toutes les class les multiples méthode qui permet de récupérer les attributs de certains objets, qui retourne des valeurs ou encore les méthode qui permettent de Set les valeurs.

- Afin d'organiser le projet il y a plusieurs packages qui ont été crées, le package Token qui contient tous les Token, le package Interface qui contient l'interface Graphique de l'application et le package AI qui contient l'algorithme et les class et méthode qui permet la résolution du programme.

C) Élément Technique

L'Algorithme AStar

• Astar ou encore A* [8] est une amélioration/optimisation d'un autre algorithme de recherche de chemin nommée l'algorithme de Dijkstra conçus par Edsger Dijkstra [9] .

• Il fut conçu par Peter E. Hart [10], Bertram Raphael [11] et Nils Nilsson [12]. Ils ont tous les trois travaillés sur l'algorithme de Dijkstra afin d'améliorer la planification de Shakey le robot [13]

• Lors de la réalisation du Projet Sokoban, nous devions concevoir Une Intelligence Artificiel (IA), pour cela, il nous étais demander d'utiliser l'algorithme de pathfinding le plus courant dans le monde du jeu vidéo qui est l'Algorithme AStar (ou noté A*), cette Algorithme utilise l'Heuristique afin de trouver le chemin le plus rapide entre deux point précis, ici l'utilisation de l'algorithme ce contenante a la réalisation d'un niveau donnée, pour cela on récupère un état du jeu, puis on l'ajoute a deux variable, ici CloseSet et OpenSet, par la suite on récupère le premier élément de l'OpenSet puis on le copie, une fois la copie effectuer on le rajoute a la CloseSet, avec la version copiée nous la recopions alors 3 fois supplémentaires et pour chaque copie du plateau nous effectuons les déplacements du joueur vers les 4 directions et nous rajoutons chaque déplacement dans l'OpenSet puis nous effectuons le même procédé pour chaque élément de l'OpenSet jusqu'à ce que L'OpenSet soit vide ou alors que l'élément sur lequel nous travaillons soit la solution du puzzle.

8. Pour plus d'information voir Rapport AStar
9. source `https://fr.wikipedia.org/wiki/Edsger_Dijkstra`
10. source `https://en.wikipedia.org/wiki/Peter_E._Hart`
11. source `https://en.wikipedia.org/wiki/Bertram_Raphael`
12. source `https://en.wikipedia.org/wiki/Nils_John_Nilsson`
13. source `https://en.wikipedia.org/wiki/Shakey_the_robot`

Code AStar

Algorithme 1 : ASTAR permet de résoudre un niveau. Première version

Entrées : un tableau de caractères jeu

Sortie : Une chaîne de caractère

```
1  openSet ← ∅
2  path ← ∅
3  closeSet ← ∅
4  openSet.ajouter(Jeu)
5  tant que NonVide(openSet) faire
6      win ← 0
7      pour  i< len(openSet) faire
8          si openSet.get(i).getF() < openset.get(win).getF() alors
9              win ← i
10         fin
11         si openSet.get(i).getF() == openSet.get(win).getF() alors
12             si openSet.get(i).getG() > openSet.get(win).getG() alors
13                 win ← i
14             fin
15         fin
16     fin
17     current ← openSet.get(win) si current end alors
18         toGoal(current)
19     fin
20     openSet.remove(current)
21     closeSet ← current
22     neighbours ← current.getNeighbours()
23     pour chaque neighbour : neighbours faire
24         si si neighbours n'est pas un mur et une zone dangereuse et qu'il n'est pas
             contenue dans closeSet alors
25             tempG ← current.getG() + 1
26             si openSet ne contient pas neighbour alors
27                 openSet.add(neighbour)
28             fin
29             neighour.setG(tempG)
30             neighour.setHeuristique(neighbour, end)
31             neighour.setF()
32             neighour.setPrevious(current)
33         fin
34     fin
35 fin
```

Algorithme 2 : ASTAR permet de resoudre un niveau. Version Final

Entrées : un tableau de caractères jeu
Sortie : Une chaine de caractère

```
1  openSet ← ∅
2  openSet.ajouter(Jeu)
3  closeSet ← ∅
4  closeSet.ajouter(Jeu)
5  tant que NonVide(openSet) faire
6      current ← openSet.poll
7      action ← current.getMaze() solution ← current.getSolution pour chaque
        d : deplacement faire
8          eltCurrent ← copy(action) eltCurrent.getPlayer().inputMove(d)
            terrain ← mazeConverter(eltCurrent) si history ne contient pas essai alors
9              newSolution ← solution + d
10             si estTermine(essai) alors
11                 this.path ← newSolution retourner NouvelleSolu
12             fin
13             openSet.ajouter(MazeModel(eltCurrent, newSolution)
14             closeSet.ajouter(terrain)
15         fin
16     fin
17 fin
```

L'Interface Graphique

Expérimentation

• Afin de vérifier l'application, nous avons effectué différent testes afin de s'assurer que le programme fonctionne : Tout d'abord nous avons effectué des tests afin de voir si on change de type de fichier cela fonctionne avec d'autre type, alors que en faite le typage de la variable est et restera de type char ou int selon la version prise , de plus nous avons effectué des tests afin de vérifier si l'on effectuer des déplacement des casses sur des murs par les points cardinaux, de même pour le personnage et essayer de déplacer plusieurs caisse d'affiler. Nous avons aussi effectuer des tests sur l'interface Graphique, nous avons essayé plusieurs Layout lors de sa création, plusieurs position et aussi image. De plus nous avons essayer de corriger toutes les erreurs humaines que l'utilisateur peut faire

III Suite Au Projet

A) Problème obtenue

Durant les 4 mois de programmation sur le projet, nous avons fait face à de nombreux problèmes.

- Pour commencer nous avons eu des problèmes avec les méthodes de déplacement, en effet durant une partie de la programmation nous ne modifions pas les positions des Token, ce qui nous faisait des erreurs comme des caisses qui ne ce déplaçaient pas ou encore le personnage qui disparaissait, c'est après quelque temps et une relecture du code et des tests effectués que nous avons rajouté les lignes afin de modifier les posX et posY des Tokens. De plus durant la réalisation du projet nous avons été ralentis par les méthodes qui empêchent le personnage de sortir du plateau et de ne pas écraser/ traverser les murs ou les Boites, pour cela nous avons créé une méthode dans les Class Player et Box, ces méthodes étant exactement les mêmes Nathan a eu l'idée de créer un classe nommée Token qui regroupe les méthodes commune entre touts les Objets qui étaient des points du plateau, du coup la class Token a été créé qui est la class mère des Box, Player et Goals. Les Déplacements fonctionnent .

- L'un des problème récurrent que Arthur a eu durant le projet est l'interface, en effet c'était la première fois que nous devions faire une interface Graphique avec Java, Arthur c'est occupé de cette partie, il a alors commencer par créer le plateau de jeux(Nathan avais fait des essais) à l'aide des dessins de Nathan, puis a commencé à créer une page d'accueil en suivant les schémas de Nathan qui lui avait passé, il a donc créé une page d'accueil avec des Boutons et une comboBox. Mais une fois cela fait, il y a eu de nombreux problèmes, en effet pendant une partie du temps, lorsque l'on cliquait sur le bouton jouer, le plateau ne s'affichait pas ou alors assez mal, Arthur a réussi à régler ce problème mais il tombât face à un autre problème lors du commencement des KeyPressed, en effet le terrain ne se mettait pas a jour car les touches ne fonctionnaient pas, les méthodes KeyPressed ne fonctionnaient pas . Pour régler tout cela il a repris le code à zéro sur une branche afin de tester les KeyPressed, une fois cela vérifié il a juste repris le code et a mis le tout dans la version principale. L'interface a été résolue

- Le problème que Nathan a eu est la réalisation de Astar (A*), en effet sur une période de presque deux mois (Voir push numéro 37 sur la forge), en effet il a commencer par ce renseigner sur les différent Algorithme de pathfinding, après s'être renseigner sur les algorithmes JPS, Breadth First Search, Dijkstra (voir Ressources/Idée/Algo.txt pour voir les liens étudier) . Après avoir lue un certain nombre de site, portfolio et visionner des vidéos sur internet, il commença a coder l'algorithme (la première version code ce trouve dans le fichier Tags/Algo et la version 2 ce trouve dans le fichier Alogo2). La première version correspond au calcule du chemin le plus rapide entre deux point pour cela l'algo utilise tous ce qui est nécessaire dans A* c'est à dire la distance entre les deux points et l'heuristique. Pour cela il créa la Class Cell qui correspond à une zone du plateau, cela pouvait être un mur ou pas, une zone dangereuse ou pas, ces voisins etc. Avec tous cela il a réussi la création de Astar mais le principe de cette Algorithme est de concevoir un sokoban solver soit un programme qui résout le labyrinthe. C'est après cela qu'il a du tout reprendre a zéros. Après un long moment, et une petite étude sur les graphes et les parcours en longueur et largeur et surtout la lecture d'un code de sokoban solver sur GitHub qu'il reprit le code et réalisa que au lieux de concevoir l'algorithme avec des Cell il faut créer des des nœuds c'est a dire mettre en mémoire un état du plateau. Il réussit cela mais avant il était bloqué avec la copie du plateau car en effet il récupérait le plateau précèdent puis

le mettra dans le nouveau plateau alors qu'il fallait le recopier . Après tout cela l'algorithme n'est pas entièrement fini car si le plateau demande énormément de nœud il y a des erreurs de Garbage Collector .

B) Amélioration possible

- Le projet, étant un projet étudiant, il ne peut qu'être amélioré, c'est pour cela que ce projet avec plus de temps de préparation pourrais avoir des modifications de notre Intelligence artificiel (IA) qui permettrait d'aider le joueur afin de résoudre le puzzle ou du moins l'aider au lieu d'une IA qui résout seul le niveau, de plus une amélioration d'A* serait envisageable afin d'empercher les erreurs de garbage collector ou encore de l'accélérer afin qu'il résolve le niveau plus vite. Il est aussi possible de revoir l'interface graphique et d'ajouter plus d'options comme un bouton pour revenir en arrière lors d'un coup rater ou plusieurs options comme mettre de la musique ou bien alors charger des niveaux fait main ou encore charger de nouvelles textures. Il serrait assez intéressent de rajouter un mod permettant au joueur d'affronter intelligence artificielle.

- Après, il est possible de rajouter plein de chose tels que la possibilité d'ajouter des musiques qui passent durant une partie ou bien des sprites différent sellons la direction choisie ou encore passer d'une interface 2D a 3D. Une des versions que nous avons prévue si on avait le temps était une version coopération ou le joueur et un ami devrons coopérer afin de résoudre le puzzle comme des caisses précise que seul deux personne peuvent pousser, de plus il était possible d'ajouter plus d'élément sur le plateau tels que des trou qui empêche le déplacement du joueur et afin de passer, il aura besoin de mettre une caisse afin de traverser ou encore des scores afin de faire concourir tous les participants.

- Le choix des modifications est tellement large que tous peut être modifié afin de le rendre le jeu/le produit plus ergonomique, plus jolie, plus pousser en terme de programmation, plus attractif, mais tous cela dépend de l'offre et la demande, en effet lorsque l'on reçoit une commande pour un projet quelconque, nous somme limiter en temps et en argent (ce qui n'importe peu dans ce projet étudiant) nous devons rendre le produit de tels sorte a ce que le cahier des charge soi respecter, nous somme développeur et non producteur et acheteur, nous devons alors respecter la demande faite par la personne.

IV Conclusion

- Effectuer un projet sur une limite de temps est assez complexe, il faut savoir s'organiser, pour pouvoir rendre le fichier comme convenu. Mais il y a de nombreuses contraintes lors de là réalisation d'un projet, nous ne sommes pas forcément au courant des méthodes à utiliser, on va devoir alors se renseigner, on peut être face a des problèmes plus ou moins complexe sur notre code. Nous devons alors faire de notre mieux afin de les résoudres pour la date limite afin de rendre un projet fonctionnel et qui respecte le cahier des charges [14]

- Durant la programmation du projet, nous avons travailler d'arrache pied afin de rendre le projet à temps.
L'application rendue n'est pas à la hauteur de nos attentes, mais nous avons essayer de faire de notre mieux. Le projet étant assez complexe (sokoban solver) Nathan n'a pas put travailler sur le code pendant un certain moment quant à Arthur l'interface Graphique avec swing lui a poser certains problèmes.

- Durant les 4 mois travailles, nous avons pu mettre en pratique les notions aborder en Complément POO. Cela nous a permis de nous imposer dans un langage que nous ne connaissons pas et cela nous a permis aussi d'apprendre a travailler avec un langage orienter objet, c'est à dire de gérer les packages afin de rendre un projet lisible et fonctionnel et gérer les différentes class pour que l'application soit conçue de manière logique.

14. cf cahier-des-charges.pdf

Glossaire

— Sokoban : Sokoban est un jeu vidéo de puzzle inventé au Japon. Cela désigne un garde d'entrepôt.

— Java :Java est un langage de programmation orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems, avec le soutien de Bill Joy (cofondateur de Sun Microsystems en 1982), présenté officiellement le 23 mai 1995 au SunWorld.

— java Swing : Swing est une bibliothèque graphique pour le langage de programmation Java, faisant partie du package Java Foundation Classes (JFC), inclus dans J2SE. Swing constitue l'une des principales évolutions apportées par Java 2 par rapport aux versions antérieures.

— Forge(informatique) : En informatique, une forge est un système de gestion de développement collaboratif de logiciel.

— Git :git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds [15], auteur du noyau Linux, et distribué selon les termes de la licence publique générale GNU version 2. En 2016, il s'agit du logiciel de gestion de versions le plus populaire qui est utilisé par plus de douze millions de personnes.

— Edsger Dijkstra : Edsger Wybe Dijkstra , né à Rotterdam le 11 mai 1930 et mort à Nuenen le 6 août 2002, est un mathématicien et informaticien néerlandais du XXe siècle. Il reçoit en 1972 le prix Turing pour ses contributions sur la science et l'art des langages de programmation et au langage Algol.

— Peter E. Hart : Peter E. Hart (né vers 1940) est un informaticien et entrepreneur américain. Il a été président et président de Ricoh Innovations, qu'il a fondé en 1997. Il a apporté des contributions significatives dans le domaine de l'informatique dans une série de publications largement citées des années 1967-1975 tout en étant associé au Centre d'Intelligence Artificielle de SRI International, laboratoire où il a également servi en tant que directeur.

— Bertram Raphael :Bertram Raphael (né en 1936) est un informaticien américain connu pour ses contributions à l'intelligence artificielle

— Nils Nilsson :Nils John Nilsson (né en 1933) est un informaticien américain. Il est l'un des chercheurs fondateurs dans la discipline de l'intelligence artificielle [la citation nécessaire] Il est le premier professeur Kumagai d'ingénierie (émérite) en informatique à l'université de Stanford, poste qu'il a tenu depuis 1990 à 1995. Il est particulièrement connu pour ses contributions à la recherche, à la planification, à la représentation des connaissances et à la robotique.

— Shakey le robot : Shakey le robot est le premier robot générique capable de raisonner sur ses actions.

— Algorithme :Un algorithme est une suite finie et non ambiguë d'opérations ou d'instructions permettant de résoudre un problème ou d'obtenir un résultat

15. source `https://fr.wikipedia.org/wiki/Linus_Torvalds`

— Astar : L'algorithme de recherche A* (qui se prononce A étoile, ou A star à l'anglaise) est un algorithme de recherche de chemin dans un graphe entre un nœud initial et un nœud final tous deux donnés. De par sa simplicité il est souvent présenté comme exemple typique d'algorithme de planification, domaine de l'intelligence artificielle. L'algorithme A* a été créé pour que la première solution trouvée soit l'une des meilleures, c'est pourquoi il est célèbre dans des applications comme les jeux vidéo privilégiant la vitesse de calcul sur l'exactitude des résultats.

— Algorithme de Dijkstra : En théorie des graphes, l'algorithme de Dijkstra sert à résoudre le problème du plus court chemin. Il permet, par exemple, de déterminer un plus court chemin pour se rendre d'une ville à une autre connaissant le réseau routier d'une région. Plus précisément, il calcule des plus courts chemins à partir d'une source dans un graphe orienté pondéré par des réels positifs. On peut aussi l'utiliser pour calculer un plus court chemin entre un sommet de départ et un sommet d'arrivée.

— A1,A2 : Version précedente de AStar (amélioration de l'algorithme de Dijkstra)

— Pathfinding : algorithme servant à trouver le chemin le plus court d'un point à un autre

— Graphe : La théorie des graphes est la discipline mathématique et informatique qui étudie les graphes, lesquels sont des modèles abstraits de dessins de réseaux reliant des objets1. Ces modèles sont constitués par la donnée de « points », appelés nœuds ou sommets (en référence aux polyèdres), et de « liens » entre ces points ; ces liens sont souvent symétriques (les graphes sont alors dits non orientés) et sont appelés des arêtes.

— Noeud : En mathématiques, et plus particulièrement en géométrie et en topologie algébrique, un nœud est un plongement d'un cercle, l'espace euclidien de dimension 3, considéré à des déformations continues près. Une différence essentielle entre les nœuds usuels et les nœuds mathématiques est que ces derniers sont fermés (sans extrémités permettant de les nouer ou de les dénouer) ; les propriétés physiques des nœuds réels, telles que la friction ou l'épaisseur des cordes, sont généralement également négligées.

— Heuristique :Au sens le plus large, l'heuristique est la psychologie de la découverte, abordée par différents mathématiciens.Au sens étroit, plus fréquent, une heuristique est une méthode de calcul qui fournit rapidement une solution réalisable, pas nécessairement optimale ou exacte, pour un problème d'optimisation difficile.