

Rapport TPA phase 1

Groupe 3 Bleu : Sokoban

Goron Nathan, De La Rosa Louis-David, Basset Emilien, Demé Quentin



Table des matières

1	Introduction	2
2	Cahier de charges	3
2.1	Déroulé fonctionnel, objectif	3
2.1.1	Phase 1 :	3
2.1.2	Phase 2 :	3
2.2	Contexte de réalisation	3
2.3	Découpage	3
2.4	Pré-requis d'utilisation	3
2.5	Règles du jeu	4
2.6	Fonctionnalités additionnelles	4
2.7	Calendrier , dates fixées :	4
2.8	Internationalité	4
2.9	Priorités, importances relatives des fonctionnalités	4
3	Détails du fonctionnement	5
3.1	Découpage et fonctionnement du moteur de jeu	5
3.2	Communication joueur-jeu , utilisation du programme	7
3.3	fonction et restriction de déplacement du personnage	7
3.4	mécanique de déplacement des caisses	7
3.5	Astar	8
4	Schéma UML	11
5	Perspectives d'améliorations en phase 2	12
6	Session de tests	13
7	État du projet en fin de phase 1 , Conclusion	14

1 Introduction

Parmi les projets proposés, nous avons fait le choix d'éviter les projets n'étant pas des jeux de peur qu'ils limitent notre créativité et notre motivation quant à son développement. Le jeu de Sokoban nous paraissait être un projet ambitieux et intéressant de par la difficulté dans l'implémentation de son IA.

Il s'agissait donc de développer un Jeu de Sokoban : Un jeu de puzzle dans lequel un personnage déplace un certains nombres de caisses sur des emplacements dans un niveau fermé, le joueur gagne la partie si il parvient à boucher tous les emplacements avec les caisses.

Après avoir songé à l'utilisation du moteur de jeu Unreal Engine, nous avons décidé de développer ce projet en Python avec l'aide de la librairie PyGame qui offre une grande liberté dans l'interface graphique ainsi que de nombreuses méthodes facilitant les mécaniques de fonctionnement de base d'un jeu.

2 Cahier de charges

2.1 Déroulé fonctionnel, objectif

Le programme sera écrit en Python à l'aide de la librairie Pygame et répondra aux critères suivants d'ici la version finale de son développement :

2.1.1 Phase 1 :

Le jeu sera jouable par humain via un prototype non-définitif(changements possibles en phase2), le joueur pourra se déplacer dans un niveau importé au format .slc composé de plusieurs éléments (expliqués dans les détails de fonctionnement du moteur de jeu).

2.1.2 Phase 2 :

Le jeu se verra ajouter une fonctionnalité de résolution automatique grâce à l'algorithme A* et aux heuristiques, cet algorithme proposera un chemin qui soit parmi les meilleurs, et ce le plus rapidement possible (dans la plupart des cas en fonction de la difficulté) pour compléter le niveau. Cet algorithme devra être anytime , c'est-à-dire capable de compléter un niveau quelque soit son point de départ. Les fonctionnalités additionnelles facultatives sont listées dans la catégorie "Fonctionnalités additionnelles".

2.2 Contexte de réalisation

Ce projet est réalisé dans le cadre de l'évaluation de la valeur "TPA" en L2 info 2016-2017 de l'université de Caen.

2.3 Découpage

Le programme sera composé des modules suivants :

- Le module sokoban qui assurera le fonctionnement général du programme (moteur de jeu, conditions de victoire, interface graphique, importations des niveaux).
- Le module classes qui comprendra toutes les classes et méthodes nécessaire au fonctionnement du jeu.
- Le module constantes stockant les importations d'images pour les personnages et les cases ainsi que les variables nécessaire pour les variations de tailles de niveaux à l'écran.
- Le module Sokobastar qui gérera le découpage pourra être modifié en fonction des choix de fonctionnalités additionnelles.

2.4 Pré-requis d'utilisation

Le jeu fonctionnera sous n'importe quel système muni de python(2.7 a 3.5) et de la librairie Pygame. Le jeu sera intuitif d'utilisation et ne nécessitera aucune connaissance informatique particulière.

2.5 Règles du jeu

Le joueur déplace un personnage dans un niveau constitué de murs, de cases vides, de caisses et de socle pour les caisses. Il faudra déplacer les caisses dans le niveau afin de couvrir tous les socles pour terminer le niveau. Le joueur ne peut que pousser et non tirer. De plus, il ne peut pousser qu'une seule caisse à la fois. Ces règles impliques que le joueur peut se trouver bloquer.

2.6 Fonctionnalités additionnelles

6 Fonctionnalité additionnelles :

Les fonctionnalités suivantes seront susceptible d'être implémentée d'ici la version finale du programme :

- Menu permettant le choix du niveau et la modifications d'éventuelles options(sons , design de la fenêtre...).
- Bouton "undo" permettant d'annuler le dernier mouvement du joueur.
- Système de sauvegarde pour conserver son avancée dans un niveau.

2.7 Calendrier , dates fixées :

Le développement du jeu est découpé en deux phases :

- Phase 1/1er semestre : réalisation d'un jeu jouable par un humain et recherches sur l'IA (A* , heuristiques), importation des niveaux.
- phase 2/2nd semestre : programmation de la résolution automatique anytime et ajout d'éventuelles fonctionnalités additionnelles.

2.8 Internationalité

Internationalité : Le jeu ne sera pas traduit et sera utilisable uniquement en Français.

2.9 Priorités, importances relatives des fonctionnalités

Le fonctionnement du jeu sans IA et la possibilité d'importation des niveaux sont prioritaires, la résolution automatique ne sera programmée qu'ensuite. Les fonctionnalités additionnelles ne seront développées qu'une fois les objectifs des deux phases remplies.

3 Détails du fonctionnement

Pour des raisons de clarté et de facilitation d'implémentation , nous avons choisi d'implémenter notre programme en orienté objet plutôt qu'en procédural.

3.1 Découpage et fonctionnement du moteur de jeu

Notre moteur de jeu repose sur 5 classes :

- Sprite
- Personnage
- Caisse
- Niveau
- LevelCollection

Les classes Sprite, Personnage et Caisse servent à représenter visuellement les objets suivants :

- Le personnage - Il représente le personnage contrôlé par le joueur
- Les caisses - Les entités manipulables par le personnage
- Les murs - Moteur de difficulté du jeu ; Ils bloquent le déplacement des caisses et du personnage
- Les cibles - Emplacements sur lesquels il faut placer les caisses
- Les cases vides - Cases sur lesquelles les personnage et caisses peuvent se déplacer

La continuité du jeu est contrôlée par la variable continuer, celle-ci est établie à 1 au lancement du jeu et à 0 lorsque le joueur complète le niveau : à ce moment , le jeu se ferme et affiche un message de victoire en console

Pour construire notre plan de jeu , nous avons choisi de décomposer notre niveau en 2 grilles nommées GameP et GameO contenant à elles deux les objets cités plus haut ; la grille gameP(game plan) contient les cibles et les espaces vides tandis que la grille gameO(game Obstacle) contient les murs, les caisses et le personnage .L'affichage du niveau consiste donc en la superposition des deux plans et permet ainsi une vérification de victoire plus facile à la fin du jeu :



FIGURE 1 – Aperçu des deux grille superposées



FIGURE 2 – Aperçu de la grille GameO

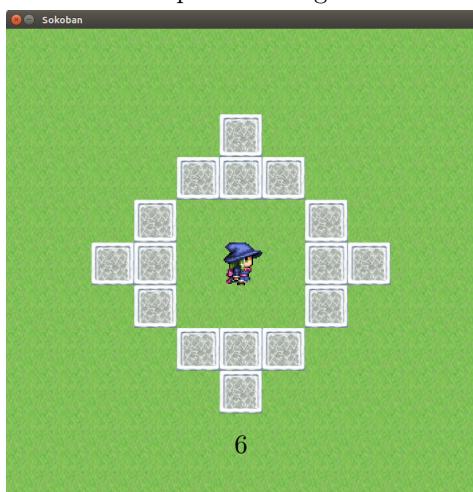


FIGURE 3 – Aperçu de la grille GameP

3.2 Communication joueur-jeu , utilisation du programme

Pour les déplacements du personnage nous utilisons la méthode event.key de PyGame qui permet de gérer des événements liés à l'utilisation du touches du clavier , 6 touches événement sont actuellement gérés : touche directionnelle droite(K-RIGHT : déplacement du personnage vers la droite) ,touche directionnelle gauche(K-LEFT : déplacement du personnage vers la gauche) ,touche directionnelle haut(K-UP : déplacement du personnage vers le haut) ,touche directionnelle bas(K-DOWN : déplacement du personnage vers le bas) , pavé numérique(KP-MULTIPLY : lance la résolution automatique du niveau avec Astar) et la touche échap(KP-ESCAPE : fermeture du jeu)

3.3 fonction et restriction de déplacement du personnage

Lorsque le joueur presse une des touche événement de déplacement citées plus ci-dessus , la méthode spéciale déplace() de la classe Personnage retournant un booléen est appelée pour vérifier si le déplacement est autorisé de la manière suivante :



3.4 mécanique de déplacement des caisses

Le déplacement des caisses est régie selon les règles suivantes :

- Le joueur ne peut pas tirer les caisses, Le joueur perd donc la partie si il pousse une caisse sur une case de coin qui n'est pas une stelle
- Le joueur ne peut pousser qu'une seule caisse à la fois

Dans le même ordre idée que pour la classe Personnage , la classe Caisse est dotée d'une méthode spéciale déplace() vérifiant si le déplacement d'une caisse est autorisé : A partir de la position du joueur et de la direction du déplacement demandé , la fonction vérifie d'abord si la prochaine case est bien une caisse et ensuite si la case suivante est une case vide ou une case stelle , si cette condition est vérifiée le déplacement de la caisse s'effectue comme exposé ci dessous :



FIGURE 4 – déplacement autorisé



FIGURE 5 – déplacement impossible

3.5 Astar

Dans notre cas l'algorithme de résolution de chemin dans laquelle une instance d'Astar est imbriquée dans une autre ; la première pour calculer le déplacement de la caisse a l'objectif , et la seconde pour calculer les déplacements nécessaire au déplacement du personnage :



Tout d'abord , Astar choisit la caisse la plus proche et calcule le chemin le plus court pour faire effectuer le premier mouvement a la caisse par le joueur



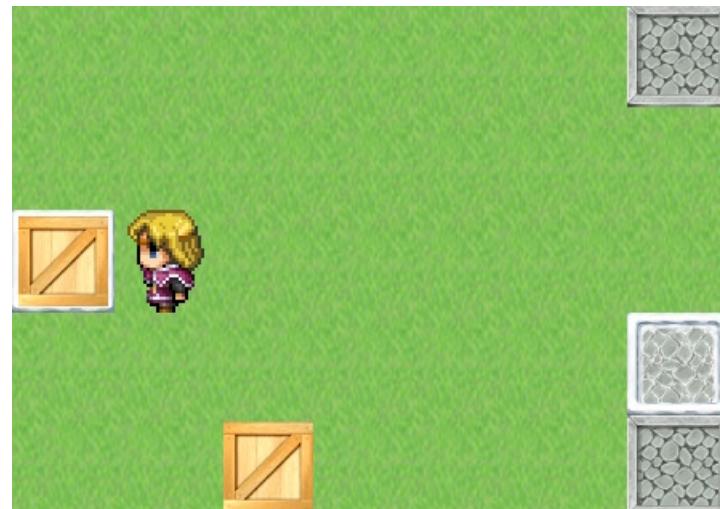
Le, joueur est positionné , Astar envoie le nombre de case que le personnage devra parcourir dans la direction dans cet exemple : 1.



Une fois la caisse poussée , Astar renvoie les nouvelles coordonnées de position du joueur pour pousser la caisse

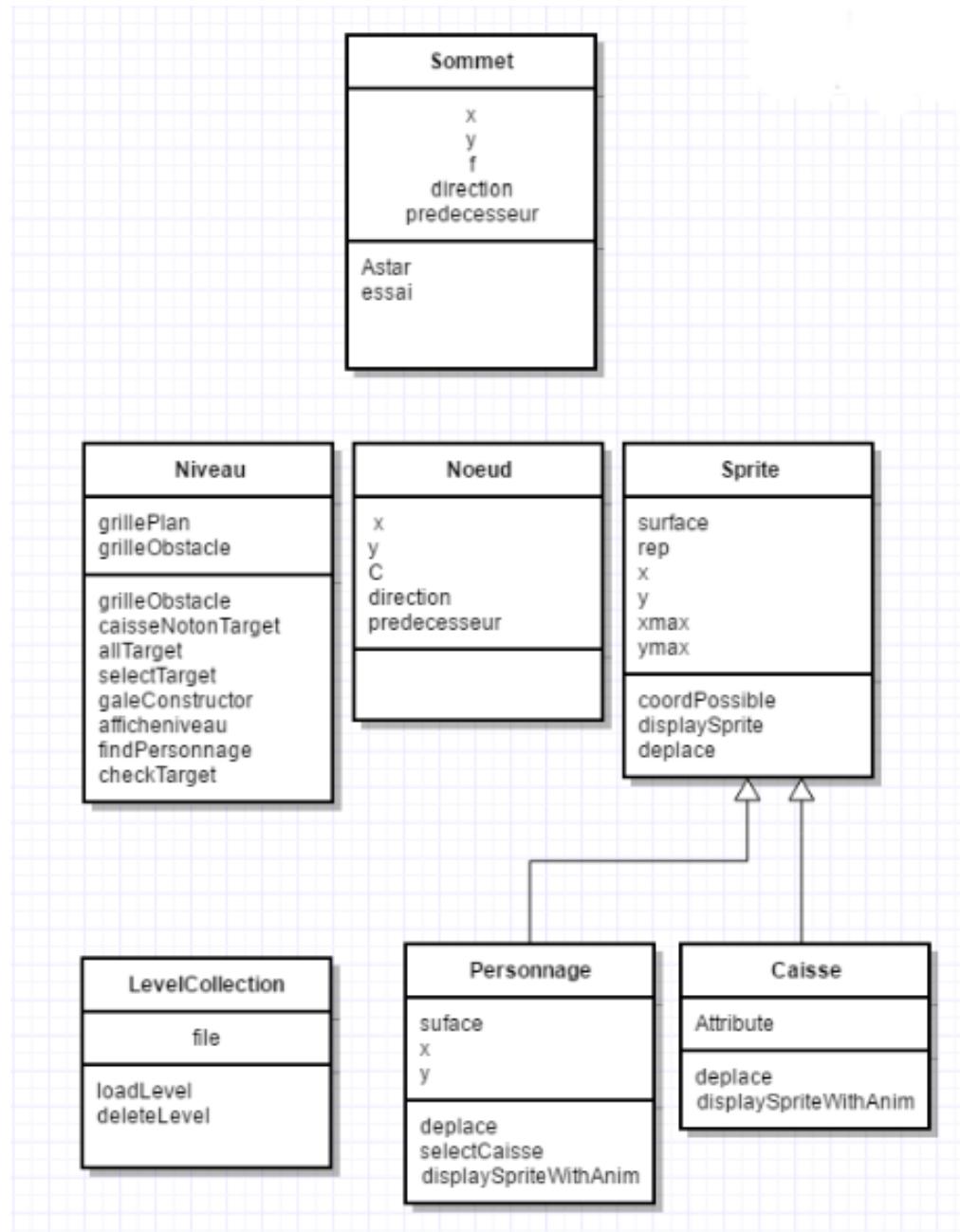


La caisse 1 est placée



La caisse sur la stelle est désormais considérée comme un obstacles au même titre qu'un mur , Astar calcule l'itinéraire vers la deuxième caisse et réitère les opérations expliquée précédemment

4 Schéma UML



5 Perspectives d'améliorations en phase 2

Après l'implémentation des heuristiques d'Astar , nous pourrons travailler sur les éventuelles possibilités d'améliorations évoquées dans le cahier des charges :

- Ajout d'un menu principal : Un menu s'affichant au lancement du jeu répertoriant plusieurs catégories ; le menu de jeu dans lequel on choisira le niveau dans lequel on souhaite jouer , un menu d'option pour régler des paramètres d'affichage ou changer le sprite de son personnage et un menu statistiques dans lequel le joueur pourra consulter ses haut-faits ou ses meilleurs scores sur les niveaux
- Sons et musique :Agrémenter le jeu d'une bande son ainsi que d'effets sonores correspondant aux actions du personnage paramétrable dans les options
- Fonctionnalité "undo" :Possibilité pour le joueur d'annuler son dernier coup si jamais il se retrouve bloqué ou souhaite optimiser son parcours
- Interface HUD :Interface en jeu donnant au joueur un certain nombre d'informations relatives à ses performances : temps de jeu depuis le lancement du niveau , nombre de coup joué , nombre de stèles restantes activer ...
- Sauvegarde de l'avancement :Pour les niveaux difficiles , le joueur pourra éventuellement sauvegarder sa partie et la reprendre plus tard grâce à un raccourci clavier ou à un bouton dans l'interface

6 Session de tests

Une batterie de tests ont été effectués via la fonction assert() pour vérifier la fonctionnalité du programme :

- tests sur classe LevelCollection
 - Aux extrêmes : grille de 9999999x9999999 -> erreur out of range
 - grille unitaire :grille 1x1 -> fonctionnel
- tests sur classe Niveau
 - test sur grille incompatible (blocs différents aux même coordonnées)
-> erreur lors des test de déplacement liés aux joueurs
 - test sur des grilles Plan et Obstacle de différentes taille -> erreur out of range si le personnage sort de la première grille
- tests sur le module Astar
 - lancement avec un nombre de caisse supérieur à celui des stelles -> arrêt lorsque plus de stelle disponible
 - test avec caisse inaccessible sur le niveau "niveautest1.slc" -> erreur
 - test sur niveau sans caisse -> retourne false(pas une erreur)
 - test sur niveau sans stelle -> retourne false (pas une erreur)

7 État du projet en fin de phase 1 , Conclusion

Au terme de cette phase 1, le moteur de jeu fonctionne. Le changement de niveau se fait via le code lui-même mais cela sera corriger lors de la phase 2. Hormis cela il est possible de jouer normalement, le programme détecte les conditions de victoire. L'algorithme A* est implémenté est fonctionnel malgré quelques bugs dans des cas spécifique ou dans les niveaux complexes.

En seconde phase , nous continuerons de développer A* afin de gérer certains cas retournant des bugs dans la version actuelle du programme .Nous commencerons ensuite à travailler sur les heuristiques d'optimisation puis sur les idées proposées en perspectives d'améliorations avec comme priorité un menu de jeu et un bouton Undo introduit dans une interface ou via un raccourci clavier.