|  |
| --- |
| Projet Sokoban |
| Projet réalisé par : Arquilliere Mathieu et Zangla Jérémy |
|  |

Table des matières

[Commande client 3](#_Toc515446155)

[Analyse de la commande 3](#_Toc515446156)

[Interface générale / cas d’utilisation 3](#_Toc515446157)

[Organisation du fonctionnement interne 3](#_Toc515446158)

[Détail de l’implémentation 4](#_Toc515446159)

[Listing 7](#_Toc515446160)

[Manuel utilisateur 7](#_Toc515446161)

[Bilan du projet 8](#_Toc515446162)

[Difficultés rencontrées 8](#_Toc515446163)

[Etat des lieux du projet 8](#_Toc515446164)

[Ressenti du projet 8](#_Toc515446165)

# Commande client

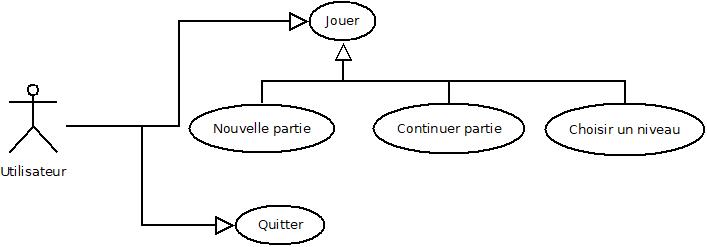
Le Joueur dirige un magasinier au sein de son entrepôt (sorte De labyrinthe qui est une grille carrée). En fait, le magasinier doit pousser des caisses afin de les amener sur des zones de rangement prédéfinies. Le Joueur peut déplacer le personnage dans les quatre directions, en général au moyen des flèches ← (4), ↑ (8), → (6) et ↓ (2). Le Personnage ne peut que pousser les caisses, à raison d’une à la fois et sous réserve qu’il n’y ait pas d’obstacle (mur, Autre caisse) ; En aucun cas, il ne peut tirer les caisses.

Le Joueur a donc pour objectif de réussir à ranger toutes les caisses aux endroits précisés avec le moins de coups possibles ; Un coup correspondant à un déplacement ou à une poussée. Une fois que c’est fait, le niveau est réussi et le joueur peut passer au niveau suivant qui est plus difficile en général. Si le joueur n’atteint pas cet objectif (caisse inatteignable, caisse bloquée), il peut recommencer le niveau en question.

# Analyse de la commande

## Interface générale / cas d’utilisation

Pour commencer le jeu, le joueur peut démarrer une nouvelle partie, qui lancera donc le niveau 1, continuer sa partie, qui reprendra au stade où le joueur a quitté précédemment, ou directement choisir le niveau à lancer.

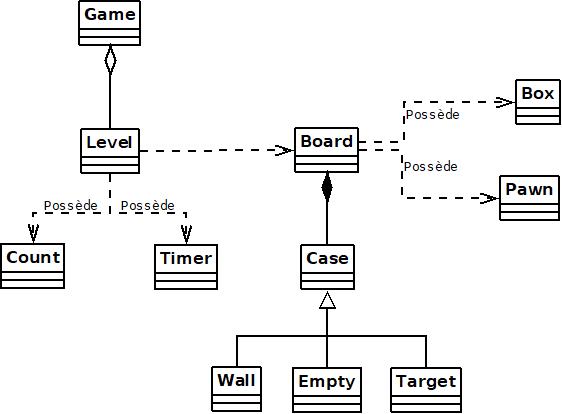


## Organisation du fonctionnement interne

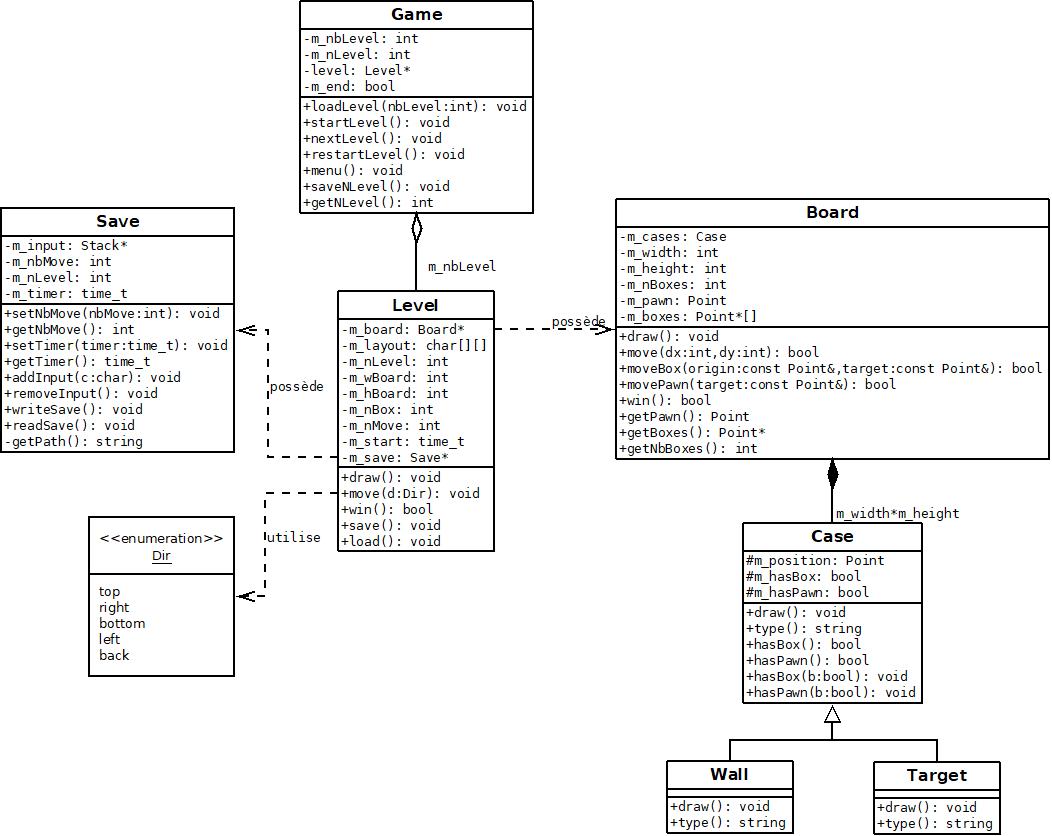
L’utilisateur doit donc avoir une interface sur laquelle il peut voir et jouer le niveau en cours. On a donc pour cela une classe Plateau (Board). Celle-ci est composée de Cases (Case) qui peuvent être une case vide (Empty), un mur (Wall) ou une case cible (Target). De plus le plateau possède un certain nombre de caisses (Box) et un pion (Pawn).

On peut donc définir ce qu’est un niveau (Level). Il possède le nombre de coups et le temps de jeu du joueur et le plateau correspondant au numéro de niveau.

Enfin le jeu (Game) se compose lui de plusieurs niveaux.



## Détail de l’implémentation



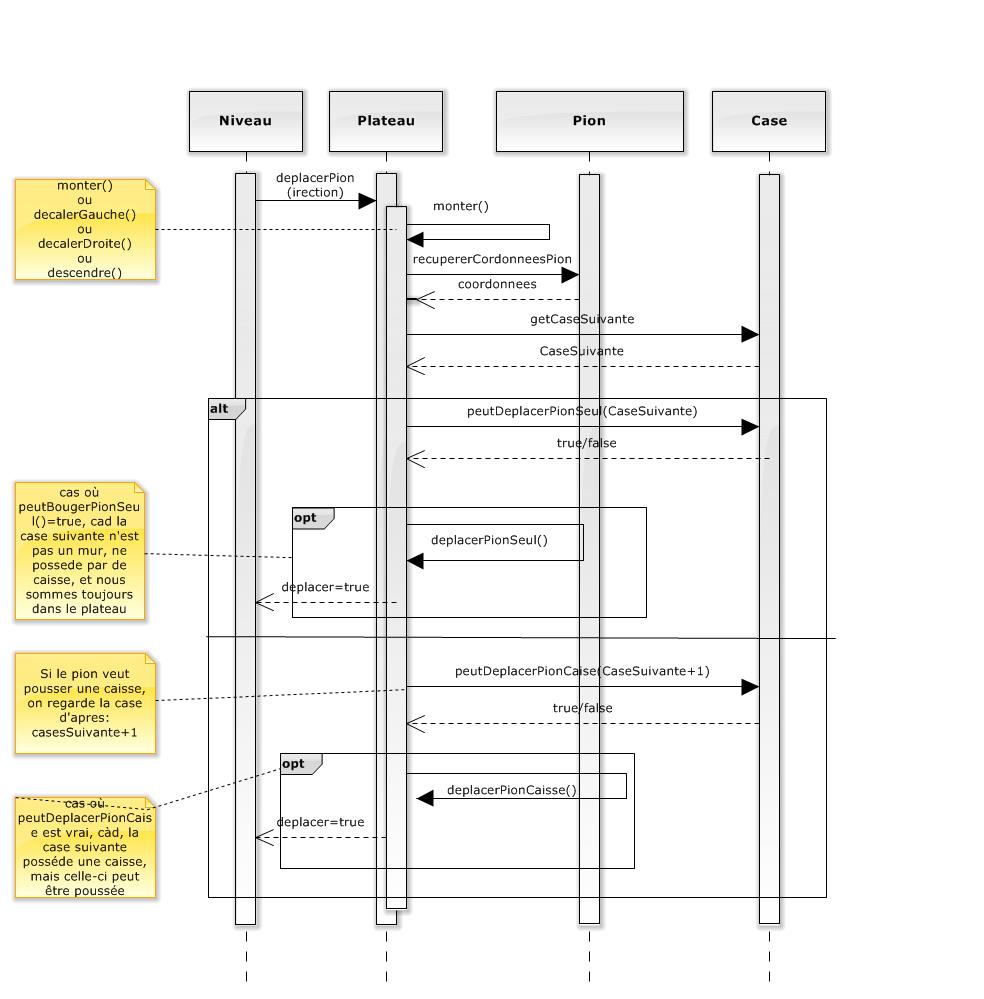
Une case par défaut est une case vide et un mur ou une case cible est une sous-classe d’une case vide. Le plateau a un tableau à deux dimensions de cases. Il a également un attribut pion qui représente les coordonnées du pion dans le tableau de cases. C’est la même chose pour les caisses (c’est un tableau de coordonnées).

Ainsi lorsqu’on veut démarrer un niveau, la classe Level trouve le fichier texte correspondant au numéro du niveau, met les caractères dans un tableau et créer un plateau avec celui-ci. Le plateau traite chaque caractère et créer une case correspondante à chaque fois. (‘#’ -> mur, ‘.’ -> case vide, ‘o’ -> case cible). De plus, il initialise ses attributs pion et caisses (‘@’ -> pion, ‘$’ -> caisse). Le niveau initialise aussi le nombre de coups et le chronomètre, qui se mettront à jour à chaque affichage du niveau.

Le niveau dispose aussi d’une nouvelle classe qui implémente une fonctionnalité rajoutée au jeu pour l’amélioration de l’expérience du joueur : la sauvegarde. Chaque niveau possède un objet de sauvegarde (Save) qui permet d’écrire et de récupérer dans un fichier les actions du joueur. Ainsi, lorsqu’on demande au niveau de charger sa sauvegarde, il récupère les actions du joueur dans le fichier, redémarre le niveau et joue toutes ces actions. Cette implémentation permet aussi d’avoir la fonctionnalité « retour arrière » autant de fois voulu (ça enlève la dernière action de la sauvegarde, redémarre le niveau et rejoue toutes les actions jouées).

La classe jeu (Game) s’occupe d’interpréter les actions de l’utilisateur et de charger les niveaux. Il possède donc un menu principal (méthode menu()) qui sélectionne le niveau à lancer (dernier niveau joué, premier niveau ou sélection faite par le joueur). Puis lorsque le niveau correspondant au choix du joueur est chargé, l’objet Game interprète les actions de mouvement du pion, retour en arrière, redémarrer le niveau, sauvegarder le niveau, charger la sauvegarde du niveau et quitter le niveau. Il affiche le niveau en cours et les possibilités d’action à chaque nouvelle action du joueur. Redémarrer le niveau recharge le fichier du niveau en cours et le recréer. Quitter ne fait que revenir au menu et sauvegarder dans un fichier le numéro du niveau en cours (pour reprendre au même niveau plus tard). L’objet Game ne fait que transmettre au niveau actuel les autres actions possibles.

Les actions pour bouger le pion sont transmise au niveau qui « convertit » la direction demandée en coordonnées relatives ((1,0) = droite, (-1,0) = gauche, (0,1) = bas, (0,-1) = haut) et demande au plateau de bouger le pion. Une série de testes est alors effectuée pour savoir si la case où l’on veut déplacer le pion est vide, s’il y a un mur ou une caisse. Si elle est vide, le déplacement est effectué (les coordonnées de l’attribut pion change), si c’est un mur on ne fait rien, si c’est une caisse on regarde la case encore d’à côté et si celle-ci est vide, alors on déplace le pion et la caisse d’une case dans la direction voulue.



Dans un niveau, à chaque nouvelle action du joueur, un test est effectué pour quitter ou non la boucle du niveau en cours. On appelle donc la méthode win() du niveau qui elle-même appelle la méthode win() du Board qui teste si toutes les coordonnées des caisses correspondent à une case cible. Si c’est le cas, la méthode renvoie ‘true’ et l’objet Game s’occupe de charger le niveau suivant.

# Listing

<listing.xlsx>

# Manuel utilisateur

Lors du démarrage du jeu, le joueur voit le menu principal. Il a plusieurs choix et doit taper le chiffre indiqué pour le valider.

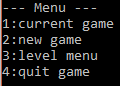
« Current game » charge le dernier niveau joué et la sauvegarde de celui-ci.

Figure : menu principal

« New game » charge le niveau 1.

« Level menu » mène au menu pour choisir le niveau souhaité (figure 2).

« Quit game » quitte simplement le jeu.

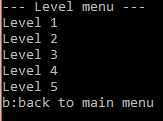
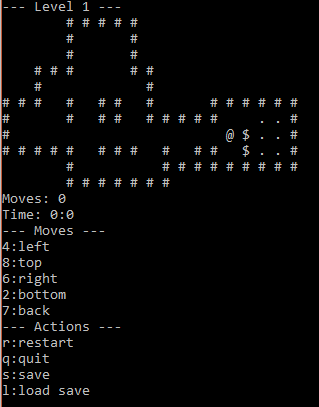
Le menu des niveaux permet de chargé le niveau souhaité en tapant le numéro de celui-ci.

Figure : menu des niveaux

Taper « b » permet de revenir au menu principal (figure 1).



Cases cibles (Zones de rangement)

Caisses

Pion du joueur

Figure 3: affichage d'un niveau

Un niveau se déroule graphiquement de cette manière : Affichage du plateau, affichage du nombre de coups et du temps, affichage des actions possibles sur le plateau et affichage des autres actions.

Le joueur peut donc bouger son pion d’une case dans la direction voulue en tapant 4, 8, 6, ou 2. Il peut revenir sur les coups faits précédemment en tapant 7.

Taper r permet de recharger le niveau, q permet de revenir au menu principal (figure 1), s permet de sauvegarder la configuration actuelle (position du pion et des caisses) et l permet de charger la dernière configuration sauvegardée.

# Bilan du projet

## Difficultés rencontrées

Lors de la création de la sauvegarde, la conception n’était pas la même. Elle s’effectuait en gardant en mémoire les positions des caisses et du joueur. Cette implémentation est sans doute moins coûteuse que la version actuelle (garder les actions du joueur et rejouer le niveau pour charger la sauvegarde) mais elle ne permet pas le retour arrière. On a donc choisi de modifier l’implémentation de la sauvegarde.

Sur ce type de projet, le travail en équipe a été compliqué à mettre en place. En effet il y a peu de code à écrire et tout est plus ou moins relié. Diviser les tâches à effectuées n’a donc pas été facile.

## Etat des lieux du projet

Actuellement le sokoban est dans une version stable qui permet de jouer 5 niveaux (de difficulté croissante), de pouvoir choisir le niveau à jouer et de reprendre là où le joueur s’est arrêté. On peut sauvegarder l’avancement dans un niveau et reprendre cette sauvegarde à n’importe quel moment. Cette version permet également un retour en arrière autant de fois qu’on le veut.

Le jeu actuel peut être amélioré sur plusieurs points. Par exemple avoir plus de niveaux, mettre à disposition du joueur un éditeur qui lui permet de créer ses propres niveaux. Une interface graphique pourrait également mieux satisfaire l’expérience du joueur.

## Ressenti du projet

Un travail agréable à mener grâce à l’apprentissage de la structuration du projet. Réfléchir sur la conception est une étape que l’on n’a peu expérimenter dans d’autre projet et le faire cette fois-ci a permis de rendre le code plus homogène, plus logique et plus agréable à réaliser. Cependant le peu de cours fait que l’on n’a pas pu approfondir ces étapes de conceptualisation afin de réfléchir à d’autre manières de réaliser ce projet.