58089 - Moufidi Ayoub (D122)
58143 - Ozkara Zeki (D122)

Baba Is You - Projet DEV4

Rapport d'analyse

Diagramme de classes

Façade

Notre modèle sera accessible depuis une façade, la classe Game. À la création de celui-ci, le premier niveau sera créé et lancé. On pourra aussi sauvegarder un niveau puis reprendre depuis la sauvegarde, ainsi que recommencer le niveau tout court. On pourra bien entendu aussi demander un déplacement pour joueur proprement dit.
Pour la plupart de ces fonctionnalités, nous avons besoin de garder en mémoire le numéro du niveau actuel, pour savoir de quel niveau on fait une sauvegarde, quel fichier à lire lors du lancement d'un niveau etc. d'où l'attribut lvlNumber.
Enfin, nous avons décidé qu'une vue pourra observer l'état actuel du niveau via une méthode getState(). La représentation choisie de l'état du niveau est la même que celle utilisée dans les fichiers de niveaux. La façon d'interpréter cet état est laissée à l'implémentation des vues. Cette méthode nous permet aussi de pouvoir sauvegarder et charger une sauvegarde facilement : il suffit d'écrire dans un fichier le résultat de getState(), puis de recréer un niveau d'après ce fichier. Comme le format est identique à celui d'un niveau de base, il n'y a aucun problème.
Pour ces opérations, il sera nécessaire de passer par une classe permettant de lire/écrire dans le système de fichiers.

Fonctionnement du modèle

Au lancement d'une partie, le constructeur de Game crée un objet Level. Cet objet est ce qui contient toutes les informations et traitements relatifs au jeu en lui-même : quels éléments y sont, quelles règles sont appliquées, etc. Le niveau prend une String dans son constructeur. Cela permet de séparer cette classe de l'aspect "lecture de fichier", et possiblement de créer un niveau depuis un texte quelconque.
Pour représenter les éléments, il faut d'abord préciser quels sont ceux-ci.

GameObject et classes dérivées

Chaque élément (non-vide) sur le "plateau" est un GameObject. Ces GameObjects ont plusieurs attributs.

- Une catégorie : une valeur d'énumération permettant de savoir si le GameObject est un élément simple (ELEM), un descripteur d'élément (TEXT) ou un aspect de règle (ASPECT). Il y a aussi le connecteur IS. Nous avons choisi de le représenter comme un élément simple, car il n'y avait pas de besoin pour une catégorie supplémentaire pour les connecteurs vu qu'il n'y en a qu'un.
- Un type : une valeur d'énumération, c'est le type d'élément tel qu'il sera réellement affiché. Par exemple, un mur sera de type WALL, un descripteur de mur sera de type TEXT_WALL, etc.

- Une direction : la direction (parmi les 4) vers laquelle cet élément est dirigé. C'est important pour les éléments qui sont affectés par la règle IS MOVE, bien que cet aspect n'est pas dans les 5 premiers niveaux.

Le type n'est là que pour faciliter l'affichage. Dire que "tels types d'éléments appartiennent à telle catégorie" aurait été préférable, mais avait l'air non seulement trop compliqué mais en plus difficilement faisable en C++, donc nous avons préféré décliner cet aspect en deux attributs.
La catégorie est particulièrement importante, car elle régit le comportement permis avec cet objet. Dû à la façon dont les énumérations fonctionnent en C++, il n'y a aucun moyen de vérifier que le type soit en concordance avec la catégorie. Cela dit, les endroits où des GameObjects seront créés sont limités et donc le risque d'entrer de mauvaises données est limité.
De cette classe GameObject dérive[1] une sous-classe : Text. Cette classe décrit les descripteurs d'éléments. Elle possède le type d'élément simple auquel elle fait référence.

Exemples

- Mur : Je crée un GameObject de catégorie ELEM avec comme type WALL.
- Descripteur de mur : Je crée un Text avec comme type TEXT_WALL, et WALL comme type de référence. Sa catégorie sera TEXT.
- Aspect KILL : Je crée un GameObject de catégorie ASPECT avec comme type KILL.

Représentation du niveau

Les niveaux ont deux informations principales à connaître : quels sont ses éléments constituants et quelles sont les règles en application.
Pour ces deux informations, nous avons choisi d'utiliser des unordered_multimap.

Éléments constituants

Une unordered_multimap<Position, GameObject>. Nous n'avions pas pu passer par un tableau 2D car plusieurs GameObjects peuvent être sur la même case, sans parler de l'espace inutile réservé pour les nombreuses cases où il n'y a pas d'éléments. Une liste n'aurait pas pu aller non plus, car la recherche n'était pas efficace. Nous avons donc opté pour une map, dont la clé serait la position, et dont les (au pluriel car c'est une multimap) valeurs seront les GameObjects à cette Position (la structure Position est une simple structure composée d'un x et d'un y).
Les avantages ? Pour l'affichage et la recherche des objets jouables, il faudra de toute façon parcourir l'entièreté du tableau, mais pour déterminer quelles règles doivent être appliquées, il faut pouvoir chercher l'objet à une position précise (aux environs d'un connecteur IS). La map nous permet d'avoir cette recherche en temps constant.

[1] On aura besoin de downcaster le GameObject pour accéder aux attributs de Text, et savoir si on PEUT le downcaster est régi uniquement par sa catégorie, qui est définissable par le constructeur. On peut mitiger ce problème en vérifiant qu'on ne peut pas entrer TEXT comme catégorie. Cela dit, il reste le fait que le downcasting n'est pas la plus élégante ni la plus efficace des façons de programmer. C'est un trade-off qui est pris consciemment dans le but de ne pas complexifier le code outre mesure.

58089 - Moufidi Ayoub (D122)
58143 - Ozkara Zeki (D122)

Règles

Une unordered_multimap<ObjectType, ObjectType[2]>. Cela nous permet facilement de déterminer quelles sont les règles appliquées pour un type d'objet donné, et d'en rajouter facilement.

Diagramme de séquences

Lorsque le joueur demande un mouvement vers la droite, cela appelle la méthode move(RIGHT) de Game qui appelle movePlayer(RIGHT) du Level en cours. Cette méthode vérifie d'abord si le niveau est déjà gagné et dans ce cas là lance une exception.

Déterminer les objets déplaçables

La première vraie tâche à effectuer est de déterminer quels sont les éléments qui vont se déplacer. Ces éléments sont ceux affectés par la règle IS YOU. Pour cela, on doit parcourir les éléments de la map contenant les règles pour trouver les clés ayant la valeur YOU[3]. Cela nous donne une liste de type d'éléments pouvant être déplacés.
Pour chaque élément de cette liste, on va chercher les GameObjects de ce type, car ce sont eux qu'il faudra réellement déplacer. On va donc, pour chaque type d'élément, parcourir les Positions occupées dans la map et chercher les éléments dont le type est celui qu'on cherche. Cela nous donne la liste des GameObjects à déplacer.

Déterminer si le mouvement est faisable

Ensuite, il faut vérifier si le mouvement est possible, et cela dépend de chaque élément déplaçable. S'il se trouve à la limite du niveau il ne pourra pas bouger, d'office, mais cela dépend souvent de ce qui se trouve à la case à côté dans la direction du mouvement.
Il faut donc déterminer quels sont les types des objets voisins et quelles sont leurs règles. Pour cela, on va d'abord chercher dans la map les types des GameObjects à la position initiale + la direction, et ensuite pour chaque type vérifier leurs règles. S'il n'y a pas d'élément à la case de destination où s'ils ne sont concernés par aucune règle, le mouvement est valide. Toutefois, si l'un des objets est concerné par l'aspect STOP, le mouvement est refusé. Si l'un des objets est concerné par l'aspect PUSH ou s'il n'est pas un élément simple, il suffira de vérifier (et pousser le cas échéant) si cet objet peut lui-même se déplacer. Le mouvement implique une suppression du GameObject à la position donnée et le déplacer vers la Position de destination.

---

[2] Il y avait une enum AspectType à la base, mais cela donnait lieu à une redondance de l'information. Des objets Aspect auraient comme type ObjectType::KILL et feraient référence à un AspectType::KILL. Nous avons donc décidé de tout rassembler dans une seule enum, ObjectType. Cela retire la protection que nous offrait AspectType dans le cadre de cette map : que pour un ObjectType, on aurait certains Aspects associés uniquement. Dans le cas actuel, rien n'empêche de dire que "BABA IS TEXT_WALL". Cela dit, ça ne devrait pas arriver programmatiquement. Les objets devraient avoir la bonne catégorie à la création. Et même si un élément normal est ajouté dans la map, il n'aura aucun effet, car l'application des règles se fait en fonction de l'aspect. Certes, la règle "BABA IS TEXT_WALL" pourrait exister, mais n'aura aucune incidence sur le jeu car ce comportement ne sera jamais défini.
[3] On doit parcourir entièrement la map. C'est dommage mais on effectue plus souvent la recherche par ObjectType.

58089 - Moufidi Ayoub (D122)
58143 - Ozkara Zeki (D122)

Construction des règles

Une fois le mouvement effectué pour chacun des objets, il faut réévaluer les règles. Pour cela, on parcourt la map à la recherche des objets dont le type est IS. Pour chacun de ces éléments, on recherche l'objet à sa gauche. S'il existe et est de catégorie TEXT, on cherche l'élément à droite. S'il existe et est de catégorie autre qu'ELEM, une règle peut être construite. Si l'élément de droite est un TEXT, il suffira de changer chaque GameObject du type de départ vers un nouveau GameObject du type d'arrivée. Si par contre c'est un ASPECT, on ajoute cet aspect à la map des règles pour l'élément décrit par le TEXT de gauche.[4]
On répète ensuite cette opération à la verticale.

Application des règles

Une fois les règles construites, on doit les appliquer. Pour cela, on vérifie chaque objet sur le "plateau" ainsi que ses règles, et on agit en fonction. S'il s'agit d'un SINK, il va vérifier s'il y a d'autres éléments que lui à la position et si oui supprime tous les éléments, dont lui-même à cette position. La règle KILL est similaire sauf que l'objet concerné est préservé et que seuls les objets affectés par YOU sont supprimés.
Enfin, il reste la règle WIN. Elle vérifie si un des éléments à la même position est de type YOU et s'il en existe, l'attribut isWon est mis à true.

L'attribut isWon

Cet attribut définit si le niveau est gagné. Si cet attribut est vrai, n'importe quel interaction de l'utilisateur ne peut se produire. Au niveau de la façade en général, après chaque mouvement cet attribut sera vérifié. Dès qu'un niveau est gagné, on charge et lance le niveau suivant.

[4] Il est inutile de garder les règles du type BABA IS WALL en mémoire, car ces règles sont appliquées une seule fois, à leur construction, puis n'affectent plus le jeu.