DEV4 rapport: UML-BabaIsYou

Dans ce rapport, nous allons expliquer en détail les diagrammes de classe et de séquence que nous avons réalisés pour modéliser le jeu Baba Is You. Le diagramme de classe représente la structure statique du système de jeu. Il montre les différentes classes du jeu telles que les objets, le board, les règles, etc … et les relations entre ces classes. Les attributs et les méthodes de chaque classe sont également représentés dans le diagramme. Nous allons décrire chacune des classes, leurs attributs et leurs relations. Par exemple, nous expliquerons comment les classes "objet" et "règle" sont reliées, et comment elles interagissent pour permettre au joueur de progresser dans le jeu. Nous expliquerons également comment les différents niveaux du jeu sont représentés dans le diagramme de classe. Le diagramme de séquence, quant à lui, décrit la dynamique du système de jeu en montrant comment les différents objets du jeu interagissent entre eux. Nous allons décrire les appels de méthode lorsque l'utilisateur effectue un déplacement à partir de l'interface graphique et comment ils sont représentés dans le diagramme de séquence. Nous montrerons comment les actions du joueur, les modifications des règles du jeu et les changements de niveaux sont représentés dans le diagramme.

Diagramme de classe

Le diagramme de classe comprend un dataType "Position" qui permet de définir les coordonnées x et y d'un objet comme "Baba", "Wall", "Rock", etc., afin de les placer sur le plateau de jeu.

Ainsi que plusieurs énumérations, l'énumération «Matériel » qui représente tous les objets du jeu, y compris les morceaux de texte qui construisent les règles du jeu, le terme 'aspect' les différencie, par exemple: "ASPECT_WALL", "ASPECT_IS" et "ASPECT_STOP" sont des morceaux de règle. En les combinant, on obtient la règle stipulant que tous les murs du jeu sont figés par exemple.

L'énumération "Aspect" représente les différentes actions qu'un objet peut effectuer dans le jeu (isYou, isPush, isStop, isWin, etc.) et l'énumération DIRECTION qui contient les différentes directions possible.

La classe "ObjectGame" représente les éléments du jeu tels que les murs, les drapeaux, Baba, le gazon, etc. Cette classe contient un attribut "Position" qui définit la position de l'objet, ainsi qu'un attribut de type "Matériel" et un attribut de type «Aspect».

La classe Level est la classe qui construit le tableau de jeu à partir d'une liste de ObjectGame. Cette liste est extraite via la méthode loadFile(int level) de la classe FileLoader qui a pour rôle de charger un fichier level, plus précisément c'est la méthode

loadFile() qui va s’occuper de retourner une liste d’objectGame contenu dans le file. Cette classe possede deux attributs board de type Board et level de type entier.

La méthode "start()" lance le jeu, elle appellera la méthode "initBoard()" qui initialisera le tableau avec toutes les cases du jeu depuis la liste retourner par loadFile(int level). La méthode "restart()" replace le plateau de jeu à son état initial et sera lancée dès que le joueur tentera de relancer sa partie.

La classe Board est une classe Observable représentant le plateau de jeu. Elle a comme attributs "height" et "width" de type entier, représentant la largeur et la hauteur du tableau. Le plateau de jeu sera un tableau dynamique (vector de vector) de type ObjectGame qu'on appellera "board". On a également un attribut "level" de type entier qui servira à passer au niveau suivant en l'incrémentant lorsque la partie sera gagnée. Cette classe comporte une liste d'ObjectGame qui ne contiendra que les "Materiel" dont l'aspect est "You", ainsi qu'une liste de règles du jeu à appliquer. La méthode "canBeMoved" permet de retourner si un mouvement est possible ou non. Cette méthode est appelée lorsque le joueur exécute un mouvement avec la méthode "changePositionObject", qui elle-même est appelée par la méthode "move".

Avant d’effectuer un deplacement la methode «changePositionObject» appelle la méthode privée isInsideRule, qui prend en paramètre le type de Matériel(getMateriel) de la case où l'on veut se déplacer et vérifie si ce type de matériel est présent dans la liste de règles si c’est le cas elle appelle la methode removeRules qui va supprimer cette regles. Apres avoir effectuer le mouvement cette méthode «changePositionObject» appelle la méthode checkBuildRule qui va vérifier si après le mouvement, une règle a été construire (explication plus détaillé plus loin chapitre diagramme de séquence).

La méthode "gameIsWin" vérifie si la position de l'objet ou des objets dont l'aspect est "YOU" se trouve à la position de l'objet ayant l'aspect "WIN". La méthode "putRules()" ajoute une règle dans la liste des règles et appel executeRule pour l’exécuter . Si une règle est brisée, elle est alors supprimée de la liste avec la méthode "removeRule(Rule rule)", qui est autorisée par la méthode "isInsideRule". Cette dernière vérifie si l'objet "YOU" pousse un morceau de règle, "isInsideRule" renvoie "true". La méthode "executeRule" parcourt la liste des règles et les exécute.

La classe Rule représente une règle, elle comporte trois attributs de type énumération "Materiel". Les attributs "subject", "is" et "aspect", en combinant ces trois attributs, on obtient une règle. Étant donnée qu’il font partie de la même énumération (Matériel) afin de différencier le «subject» et l’« aspect » la classe Board contient en attribut une liste «subjectRule»avec tous les subject (sujet possible) et une autre liste avec tous les aspect(action possible).

La classe "Facade" fournit une interface simplifiée pour interagir avec un ensemble complexe de classes afin de masquer la complexité de l'implémentation et de fournir une interface utilisateur plus simple. Dans notre cas la facade est la classe Level, elle contient les attributs des classes "Board", "ObjectGame" afin de fournir leurs méthodes au

"Controller". Le rôle du "Controller" est de recevoir les entrées de l'utilisateur, de traiter ces entrées et de déclencher les modifications correspondantes sur le modèle sous-jacent. Le "Controller" est également responsable de mettre à jour la "View" pour refléter les changements dans le modèle. Il possède un attribut "facade" de type "Facade" et un attribut "view" de type "View" pour simplifier l'interaction entre le "Controller", le modèle et la "View". La méthode "game()" construit le jeu jusqu'à sa fin. Le rôle de la "View" est de fournir une représentation instantanée du jeu en console. Elle implémente l'interface "Observer" et possède un attribut "facade" et un attribut "controller" pour interagir entre le "Controller" et le modèle. La méthode "displayBoard()" affiche le plateau de jeu et la méthode "displayWin()" affiche une interface lorsque le joueur a fini le jeu. Enfin, la méthode "askMovement()" demande à l'utilisateur d'entrer un mouvement à exécuter, ces méthode seront appeler dans la méthode update,

Diagramme de sequence

Dans cette partie, nous allons aborder le diagramme de séquence qui explique les différentes interactions entre l'utilisateur et le système, selon un ordre chronologique, lors de l'exécution d'un mouvement depuis l'interface graphique. Le mouvement est entré par l'utilisateur via la méthode askMovement, située dans la vue. Ensuite, le controleur appelle la methode move qui va appele la méthode canBeMove pour vérifié si le "You", c'est-à-dire l'élément contrôlé par le joueur, peut effectuer le mouvement souhaité. Si le mouvement est correct, canBeMove renvoie "true" et la méthode changePositionObject est appelé, elle va se charger de changer la position de l'élément contrôlé par le joueur avec la nouvelle position entrée. Sinon, l'utilisateur est invité à entrer une autre position. Si canBeMove renvoie "true", c'est-à-dire que le mouvement est possible, alors changePositionObject est appelée. Avant d'effectuer le mouvement, changePositionObject appelle la méthode privée isInsideRule, qui prend en paramètre le type de matériel (enum) de la case où l'on veut se déplacer et vérifie si ce type de matériel est présent dans la liste de règles. Si c'est le cas, elle appelle la méthode privée removeRule et lui passe l'objet contenu dans la liste de règles pour qu'elle le supprime de celle-ci et applique la suppression sur les objets concernés. Après avoir effectué le mouvement, la méthode changePositionObject va appeler la méthode privée checkBuildRule qui va parcourir le plateau et à chaque rencontre avec un objet de type "Matériel Aspect_is", vérifier si le Materiel de l'objet du jeu se trouvant à gauche ou en haut de celui-ci est contenu dans la liste "Subject", et de même pour les objets de jeu à droite ou en bas du "is", vérifier s'ils sont contenus dans la liste "Aspect". Si c'est le cas et que cette règle n'existe pas, cela signifie qu'une règle a été créée. Une instance de la classe "rule" est donc créée et passée en paramètre à "putRule", qui va ajouter la règle dans la liste de règles et appeler la méthode privée "executeRule", qui va appliquer toutes les règles contenues dans la liste de règles.