 



Class Position :

La classe Position a 2 attributs x et y qu’ils rapprissent le coordonnées de les Briques.

Fonctions :

getPosition : il return la Position a les coordonnées données en paramètre.

isOccupied : return un Boolean si la Position est occupée.

setPosition : set la Position a les cordonnées données en paramètre.

Class Bag :

La bag est unique dans le jeu. Elle contient toutes les briques.

Ella a une liste de brique et un instance de lui-même.

Fonctions :

getInstance : return la bag.

getRandomBrique: return a brique pris au hasard.

Class Brique :

Un Brique est un vecteur des Positions et contient aussi une Position central avec la quelle on va construit les autres Positon au tour de celui-là.

Fonctions :

Drop : il permet de vite déplace une brique a la fin de la Board.

checkCollision : regarde si la brique entre en collision avec autre brique déjà positionnées sur la Board.

getShape :return le vecteur des positon avec le brique a été composée.

Move : move la brique vers la direction donnes en paramètre.

Rotation : tourne la brique vers une rotation horaire or anti horaire.

Class Board :

Un board a une taille de 20\*10 et elle est compose da un vecteur de vecteur de shapes.

Fonction :

IsLigneFUll :regarde si une ligne du Board est totalement remplie .

deleteLigne : permet d’efface les case pleine d’une ligne.

moveBriqueOnBoard : déplace la brique en paramètre.

addBriqueOnBoard : il ajoute un brique sur le board.

Class Level :

La classe Level a un attribut Time de type Timer que qu’il représente la vitesse a un level donne et chaque level a un time par défaut.

Class Timer :

Permet de changer la vitesse à chaque Level.

Class Game :

La classe game est la classe principale de la modélisation du notre projet de Tetris .

Attributes :

State : est le state du game (play ,pause) ;

Level : est le level du game et si le player a obtenu suffisamment des poits il passe au niveau suivant.

Score : est le score du player

Board : est le board du game.

Bag : est le bag du game.

Brique\_courant : est le brique courant qui tombe haute vers le bas.

Fonctions :

isGameover : regarde si le player a perdu le jeu ( un certain score a été atteint, le jeu s’arrête quand un certain nombre de lignes ont été effectuées, un certain temps s’est écoulé depuis le début du jeu)

pause : il pause le jeu.

Play : mettre en action le jeu .

levelUp : si le joueur a obtenu en certain points le game passe au niveau suivant.

addBriqueTosequence : ça il nous permet de ajouter le prochaine brique a la séquence(le prochaine brique à tomber).

Drop : il il permet de vite déplace le brique courant à la fin de la Board.

updateScore : est la mise a jour du score en case d’un ou plusieurs ligne effacée.

Move : déplace le brique courante dans la direction donne en paramètre.

Rotation : tourne le brique dans le sens donne en paramètre.

Choix de la représentation d'une brique :

Nous avons opté pour la représentation d'une brique sous forme d'un vecteur de positions, où chaque position peut être soit vraie (occupée par la brique) ou fausse (libre). Bien que cette approche génère de nombreuses instances de positions et nécessite une gestion manuelle de la mémoire en C++, elle offre une grande simplification pour la modification des paramètres du jeu (comme par exemple faire un mouvement de brique).

La deuxième option envisagée consistait à utiliser des matrices de booléens pour représenter une brique. Cependant, nous avons constaté que cette approche serait plus complexe à concevoir et à coder. Bien que l'utilisation de cette manière de faire aurait permis de réduire le nombre d'instances de positions, cela aurait entraîné une confusion lors de la conception du code.

Choix de l'héritage pour les briques :

Nous avons choisi de définir les briques en tant qu'enfants d'une classe parent plutôt que d'opter pour une classe factory. Bien que cela entraîne la création de nombreuses classes enfant, cette approche offre une protection accrue des méthode spécifiques à chaque type de brique. Ainsi, nous avons la certitude qu'une brique, telle que la brique L, est explicitement une brique de type L avec des positions vérifier et précise. Cette spécificité est assurée par les attributs de la classe, notamment une liste de positions qui dessine de manière précise la forme L, éliminant ainsi toute ambiguïté.

Choix de la classe principale du modèle :

La classe Game a été choisie comme classe principale afin de centraliser l'ensemble des méthodes et d'offrir un accès complet au modèle. Cette approche s'apparente au design pattern façade, permettant au contrôleur d'interagir uniquement avec cette classe plutôt que de naviguer à travers l'ensemble des classes du modèle. Cette centralisation simplifie la maintenance et l'extensibilité du code.

Choix de la relation entre Time et Level :

Nous avons décidé de lier les classes Time et Level, car nous anticipons que le niveau affectera la difficulté du jeu en modifiant le timing. Cette liaison facilite les appels entre les deux classes et garantit que Time n'est pas modifié par d'autres classes de manière inattendue. En unissant ces deux composantes, nous simplifions la gestion des paramètres liés au temps et à la difficulté du jeu.

Choix des vecteurs:

On a choisi d'utiliser des vecteurs car la taille est dynamique, et donc on pourra mettre autant de brique dans le vecteurs que l'on veut car les vecteurs peuvent se redimensionner mais le redimensionnement va entrainer une mauvaise gestion de la mémoire et le déplacement des éléments existant. De plus on a un accès direct aux l’éléments du vecteur. Mais comme dans la mémoire les objets sont mis de manière contiguë ce sera moins couteux de faire des itérations et donc on a une utilisation plus rapide de la mémoire.