Règle du jeu Corridor+

Taille du plateau : Modifiable, taille de base 9x9

Principe général :

Les deux joueurs choisissent un pion parmi plusieurs disponibles au début de la partie, ainsi qu’un ensemble de mur à disposition. Le but est d’atteindre la ligne de départ de l’adversaire.

Décomposition d’un tour :

Chaque joueur joue une action par tour. Au choix :

* Poser un mur
* Déplacer une unité
* Utiliser un pouvoir

Déplacement des unités :

Déplacement d’une case, pas en diagonal, impossible de traverser les murs. Si un pion A colle un pion B, le pion A peut « sauter » par-dessus le pion B, a la condition que la case immédiatement derrière soit disponible.

Poser un mur :

Choix parmi un ensemble de mur, on ne peut ni enfermer un pion ni couper un autre mur.

## Moteur du jeu

## L’initialisation

La phase d’initialisation est l’occasion de choisir le type de pion qu’on veut jouer, de le déployer sur le labyrinthe, et de choisir l’ensemble des murs à disposition pour la partie.

1. Vous choisissez une unité parmi les pions disponibles. Vous positionnez cette unité sur la première ligne de votre côté, au milieu de la ligne.
2. Vous choisissez l’ensemble des murs disponibles par un système de point.
3. Un premier joueur est déterminé.

## Un tour se déroule comme suit :

Vous avez le choix entre l’une des trois actions suivantes :

1. Déplacer une unité
2. Positionner un mur
3. Utiliser un pouvoir

## Communication avec le serveur

<https://github.com/Rakhx/L3ProjetLong.git>

Le faux client avec le prototype des fonctions à appeler est disponible sur le github ci-dessus, et sera mis a jour au fur et a mesure des séances de TP.

Des détails sont disponibles dans le fichier Python du client.

# Plateau de jeu

Le plateau de jeu est composé de cases pour positionner les unités, ainsi que de cases pour positionner les murs.

Le plateau de jeu sera un plateau de taille double, ou chaque ligne et colonne entre les cases de pion seront des cases de mur.

Une image contenant jouet, Modèle réduit, Brique de construction, intérieur

Description générée automatiquement

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

En blanc, les cases pour les unités.

En gris les cases pour les murs

En noir les cases non utilisables

La taille du plateau est donc doublée pour pouvoir prendre en compte les positions possibles pour les murs. On défini que le point (x=0, y=0) se trouve

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 : 0 |  |  |  |  |  |
| 1 : 0 |  |  |  |  |  |
| 2 : 0 |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# Unités

Au début du jeu les joueurs choisissent un pion parmi plusieurs disponibles :

* pionSappeur : Unité qui casse un mur au contact, l’ensemble du mur.
* pionSprinter : Unité qui peut se déplacer de deux cases
* pionJump : Unité qui peut sauter par-dessus un mur

Nous commençons par un temps de rechargement de chaque pouvoir de 3 tours, valeur qui peut être amenée à changer pour équilibrage. (Si vous utilisez votre pouvoir au tour 1, il sera à nouveau disponible à votre tour 4 de jeu)

# Mur

Chaque mur va couter un certain nombre de points. A l’initialisation, le joueur dispose d’un nombre de piece d’or (PO) donné et pourra choisir les murs qu’il souhaite pouvoir utiliser.

Type de mur

* WallSolide – 2 PO - Mur incassable et insautable
* WallLong – 3 PO - Mur de longueur 4 cases
* WallTempory – 1 PO Mur temporaire (3 tours). Les murs temporaires disparaissent au début de votre tour
* WallDoor – 3 PO Mur qui peut être traversé que par le joueur qui l’a posé
* WallClassic – 1 PO

|  |  |  |  |
| --- | --- | --- | --- |
| Numero du mur | Nom du mur | Cout du mur |  |
| 0 | Wall Classique | 1 |  |
| 1 | Wall Solide | 2 |  |
| 2 | Wall long | 3 |  |
| 3 | Wall Door | 3 |  |
| 4 | Wall Tempory | 1 |  |

A l’exception du murLong, les murs ont une taille de 2 cases // briques. Chaque mur est caractérisé par un point de départ, une direction ( haut, bas, droite, gauche) et une longueur.

On choisi de donner 15 PO à chaque joueur, valeur qui peut être équilibré par la suite.

On place les murs par une position de départ et une orientation. L’orientation tourne dans le sens horaire ( comme une montre), en commençant en haut par 0.

|  |  |
| --- | --- |
| 0 | Haut |
| 1 | Droite |
| 2 | Bas |
| 3 | Gauche |

# Le modèle Vue Présenter & le fonctionnement du jeu

Pour faire marcher votre boucle de jeu, vous pouvez faire un main qui import model, view, présenter. Il les relie ensemble, puis demandera au model de commencer la boucle de jeu.

Les fonctions du serveur ne laissent pas de place aux erreurs. Si une requête est incorrecte, un comportement par défaut est mis en place, et les procédures de jeu continuent.

Les étapes pour commencer à jouer sont les suivantes :

* Appelle de la fonction registerTeam avec le pion choisi et le nom de l’équipe. Comportement par défaut : Nom d’équipe : player1, player2, et pion par défaut : pion sprinter.
* Appelle de la fonction choixMur, qui contient l’ensemble des murs acheté par le joueur. Le parametre est une liste d’entier, représentant les murs choisis. Comportement par défaut : Que des murs classiques.

Ensuite, des fonctions sont à appeler régulièrement pour faire fonctionner les boucles de jeu.

* Afin de gérer le tour par tour, il est nécessaire de commencer la boucle de jeu par l’appel à la fonction askPriority(). Cette fonction est bloquante, c’est-à-dire que votre programme sera bloqué dans cette fonction tant que ce n’est pas à votre tour de jouer. Votre programme sera donc en « pause ». Cette fonction renvoie l’état de la partie sous la forme d’une matrice (à définir plus précisément), qui représente l’état de la partie :
  + La position des pions sur le plateau
  + La position des murs sur le plateau

La forme sera probablement une liste de liste rempli de caractere, chaque caractere représentera un type d’élément sur le plateau.

Une fois cette fonction terminée, vous pouvez prendre votre tour

* Vous pouvez ensuite appeler une fonction parmi les trois disponibles
  + Déplacement. Cette fonction vous permet de déplacer votre pion dans une direction donnée. Comportement par défaut : pas de déplacement
  + placementMur Ce fonction permet de choisir un mur à placer et de le mettre sur la grille. Les paramètres sont le type de mur (numéro, voir plus haut) la position du début du mur (deux coordonnées, X et Y), et une orientation ( entre 0 et 3) . Comportement par défaut : pas de placement de mur, et le mur n’est pas utilisé.
  + UtilisationPouvoir. On active un pouvoir, et choisi une direction pour le sens d’application. Le jumper se déplace donc de deux places dans la direction données, le sauteur se déplace, si le mur n’est pas un mur infranchissable, et le sappeur détruit le mur s’il existe et se déplace. comportement par défaut Déplacement dans l’orientation choisi.

# ~~Evaluation~~

~~L’évaluation sur le projet pour le premier semestre va prendre en compte des éléments à plusieurs niveaux :~~

* ~~L’existence d’une architecture de classe qui permettra une évolution facile pour le passage au deuxième semestre (a minima une séparation des fonctionnalités et des références limités à d’autres classes)~~
* ~~L’utilisation de classe, et le respect des paradigmes de la programmation objet (encapsulation, héritage lorsque c’est pertinent, surcharge des opérateurs & fonctions…)~~
* ~~Niveau du code Python (fonction avec paramètre optionnel, boucle claire et optimale…)~~
* ~~La lisibilité du code : nom de variable explicite, commentaire en début de fonction pour expliquer son rôle et son utilisation~~

~~La date finale de rendu est fixée au 7 janvier 2023.~~

## Objectif du projet pour le premier semestre

L’application du premier semestre aura une **view** passant par la console, et un **model** minimal. Ces deux éléments évolueront ensuite pour intégrer une interface graphique et un **model** plus complexe.

## Concernant l’architecture des classes

L’objectif du travail du premier semestre est de réaliser une application « minimale » qui sera améliorée au second semestre. Un soin particulier va donc devoir être pris pour la définition de l’architecture des classes. Il est demandé de suivre le patron de conception **Modèle Vue Presenteur** ( MVP – **Model View Presenter**) vu en cours afin de séparer proprement les trois fonctionnalités. Chacune de ces fonctionnalités peut être constitué de plusieurs classes.

Dans un 2eme temps, il peut être envisager d’utiliser des classes abstraites pour définir les fonctionnalités minimales du model, de la **View** et du **Presenter**. Des classes de **view**, **model** et **presenter** concrète seront donc créés en héritant de leur contrepartie abstraite.

## Remarques générales

Certaines des fonctionnalités qui seront nécessaires au fonctionnement du jeu ne sont pas décrites dans le document. A vous de faire des choix sur leurs implémentations pendant votre implémentation.

Pensez le plus possible en terme factorisation de code : si un « bout » de code est utilisé à plusieurs endroits, envisagez d’en faire une fonction. Lorsque vous écrivez une fonction, essayez de la rendre la plus générique possible. Envisagez que les fichiers à charger puissent évoluer, que la taille du terrain aussi, etc.

Si des classes partages certaines propriétés et fonctions en communs, pensez à l’héritage