Règle du jeu Corridor+

Taille du plateau : Modifiable

Principe général :

Plateau taille X taille.

Les deux joueurs choisissent un pion parmi plusieurs disponibles au début de la partie. 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 de 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.

Poser un mur :

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

Demande du premier semestre

Votre bot joue seul contre un bot qui se déplace aléatoirement avec une pondération pour la direction de votre plateau.

Vue console pour le premier semestre.

## 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.
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>

Au second semestre, le projet s’orientera vers une utilisation client/serveur. Afin de se préparer à ça, un faux client sera disponible au premier semestre pour faciliter le passage au second.

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

# Unités

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

* pionSappeur : Unité qui casse un mur au contact
* pionSprinter : Unité qui peut se déplacer de deux cases
* pionJump : Unité qui peut sauter par-dessus un mur

Chacun de ces pouvoirs a un temps de rechargement, à déterminer.

Une unité possède une position.

# Mur

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

Type de mur

* WallSolide Mur incassable
* WallLong Mur de différente longueur ( 2, et 4 )
* WallReusable Mur réutilisable ( pick and redeploy
* WallTempory Mur temporaire ( X tour )
* WallDoor Mur // porte

Chaque mur est caractérisé par un point de départ, une direction ( haut, bas, droite, gauche) et une taille.

## Le plateau de jeu

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.

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

Partie **view** :

Trouver une manière d’afficher dans la console le terrain depuis ses dimensions (ou depuis sa représentation) dans le **model**. N’oubliez pas que le **presenter** sera la au besoin pour effectuer des transformations des données côté **model** pour l’affichage côté **view**.

Quelques idées (pas forcément pertinentes) de ce qui peut être fait en affichage console : <https://www.asciiart.eu/>

# 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