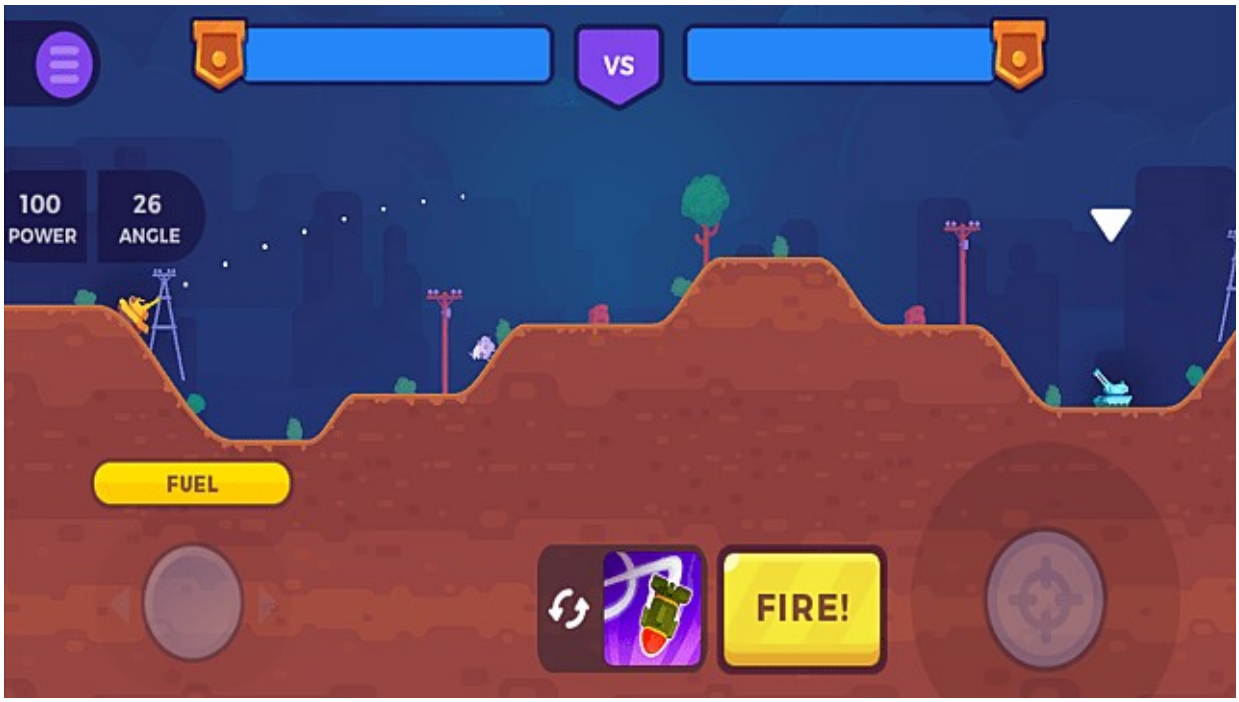
Projet Logiciel Transversal

Eddy LOBJOIS – Shuman LIN – Siaka OUATTARA

Table des matières

1 Présentation Générale

1.1 Archétype



Le projet de ce semestre va s'appuyer sur le jeu Tank Stars. Il s'agit d'un jeu avec des mécanismes de gameplay similaires à Worms, mais où les vers de terre ont été remplacés par des tanks.
Dans ce jeu, deux tanks s'affrontent tour par tour dans une bataille dont le but est de détruire le tank adverse.

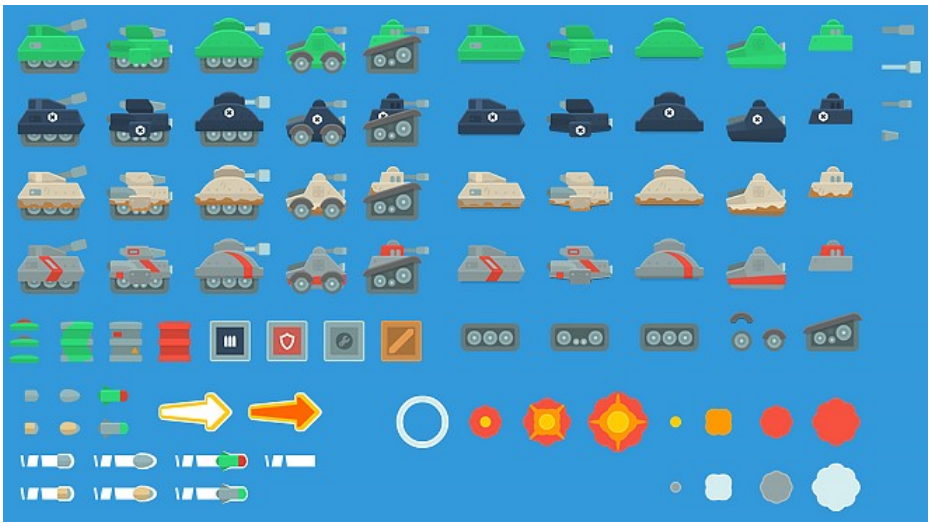
1.2 Règles du jeu

C'est un jeu de combat au tour par tour à deux joueurs.
- Au début du jeu, chaque joueur doit choisir un tank parmi deux modèles disponibles : Le premier dispose de 80 points de vie (PV) et de 20 points d'attaque (PA). Le second dispose de 100 PV et de 16 PA.
- Une fois que les deux joueurs ont choisi le tank, les points de vie sont initialisés.
- Durant chaque tour, le Tank attaquant peut se déplacer de gauche à droite. Il peut également ajuster la trajectoire angulaire et modifier la puissance du tir.
- Au bout de 30 secondes, les prédispositions doivent être effectuées, le tank d'attaque va lancer le tir de missile suivant une trajectoire oblique.
- Si le tir touche le tank défensif, son nombre de points de vie va diminuer par rapport aux PA d'adversaire.
- Quand le tir de missile est terminé, les 2 joueurs changent leurs rôles.

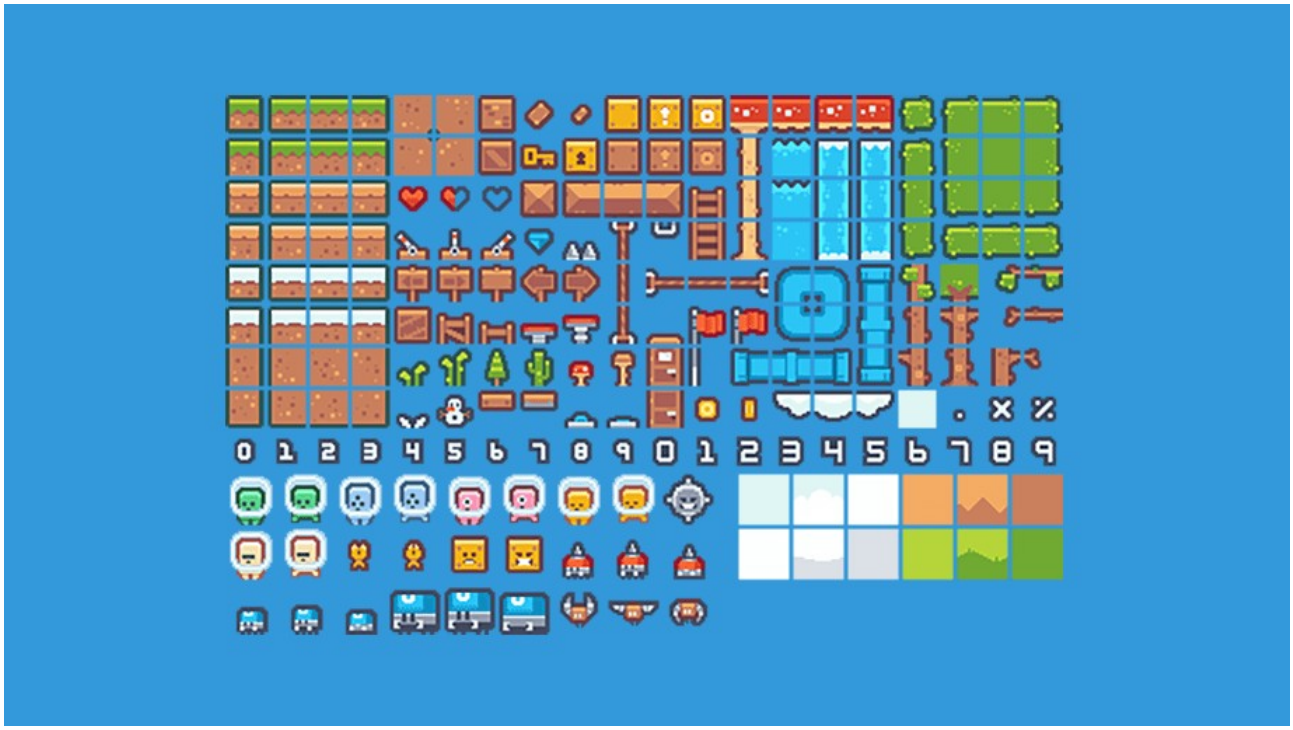
- Si le nombre de points de vie d’un des deux tanks arrive à zéro, la partie se termine et le tank encore en vie remporte la partie.

1.3 Conception Logiciel

Présenter ici les packages de votre solution, ainsi que leurs dépendances.



Bibliothèque de sprites utilisés (source: kenney.nl)



Bibliothèque d’éléments de décor (ou tiles) utilisés (source: kenney.nl)

Pour l’affichage d’une map complète, nous utiliserons un vecteur constitué de 20 tiles de longeur et de 15 tiles de hauteur juxtaposés, puis superposés à l’écran. Chaque tile est modélisé par une carré de 36 pixels de côté, ce qui permet d’obtenir une map d’une taille équivalente à 720 par 540 pixels.

2 Description et conception des états

2.1 Description des états

Cases « Tank ». Cet élément est dirigé par le joueur, qui commande les sens à droite et à gauche. Pacman dispose également d’un « compteur super », qui sert à déterminer le temps restant avant de repasser en normal. Enfin, on utilise une propriété que l’on nommera « status », et qui peut prendre les valeurs suivantes :
— Status « Attack » : cas où le tank peut se déplacer sur la pelouse et ajuster l'angle de cannon.
— Status « Defend » : cas où le tank peut pas bouger et attend que l'adversaire de finir sa tour.
— Status « Mort » : cas où le nombre de points de vie de tank soit à 0.

Cases « Cannon ».
Cases « Weapon ».

Cases « Panel ».

Cases « Pelouse ». Les cases « Pelouse » sont les éléments sur lesquelles les tank peuvent se deplacer.

Cases « Obstacle ». Les cases « Obstacle » sont des éléments infranchissable pour les tanks. On considère les types de cases « Decoration » suivants :
— Les espaces « riviere», qui se trouve au millieu de la scene du jeu
— Les espaces « rive», qui se trouve au borde de la riviere pour empecher les tanks de tomper dedans

Cases « Decoration ». Les cases « Decoration » sont des éléments intouchable pour les tanks. On considère les types de cases « Decoration » suivants :
— Les espaces « ciel »
— Les espaces « nuage »

*Pour les cases « Obstacle » et « Decoration », le choix de la texture est purement esthétique, et n'a pas d'influence sur l'évolution du jeu.

Quatre états de jeu sont possibles:
GAMEOVER: le jeu est fini.
AVAILABLE: le joueur principal peut se déplacer, ajuster la trajectoire angulaire de canon et modifier la puissance de son tir.
SHOOTING: le joueur principal ne peut plus se déplacer, son tir est lancé en suivant une trajectoire oblique.
AWAITING: le joueur principal attend que le joueur adverse joue son tour.

Chaque élément de la map autorise ou non le déplacement d'un tank de manière à ce que ce dernier puisse uniquement se déplacer sur des blocs de type «verdure»
Les macros blocType sont les suivants:
- EMPTY: le tank ne peut pas se déplacer
- SOLID: le tank peut se déplacer à gauche et à droite
- LEFT_BORDER: le tank peut uniquement se déplacer à droite
- RIGHT_BORDER: le tank peut uniquement se déplacer à gauche

2.2 Conception logiciel

Les principaux éléments et états du jeu sont présentés dans une classe **State** contenant:
- Une map modélisée par un vecteur de nombres entiers où chaque indice est associé à une partie découpée d'un tileset (peut être vide ou de type verdure, bordure, étang…)
- Deux joueurs (le nôtre et l'adverse) possédant chacun leurs caractéristiques relatifs aux tanks, au tirs, aux positions et angles.
- Un attibut **turnID** qui détermine le joueur qui est autorisé à jouer le tour à l'instant présent.

2.3 Conception logiciel : extension pour le rendu

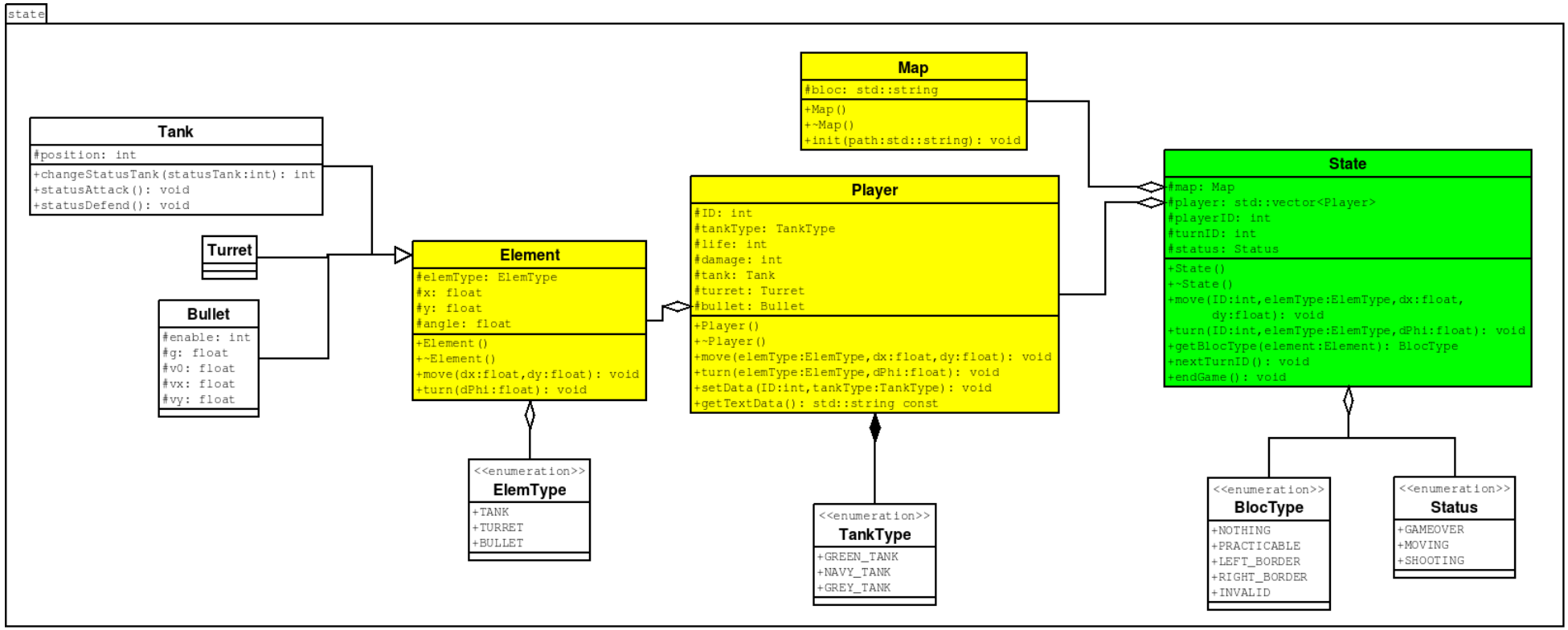
La classe **Scene** est constituée d'une fenêtre de rendu **window**, d'une machine à état et d'une classe dérivée **Surface** disposant des méthodes suivantes:
- Chargement d'éléments graphiques tels qu'une map, un sprite ou un panneau.
- Affichage d'éléments graphiques multiples rendu possible par l'utilisation une unique méthode abstraire **draw**.

2.4 Conception logiciel : extension pour le moteur de jeu

2.5 Ressources

Tileset utilisé pour la composition de la map

Illustration 1: Diagramme des classes d'état



3 Rendu : Stratégie et Conception

Présentez ici la stratégie générale que vous comptez suivre pour rendre un état. Cela doit tenir compte des problématiques de synchronisation entre les changements d'états et la vitesse d'affichage à l'écran. Puis, lorsque vous serez rendu à la partie client/serveur, expliquez comment vous aller gérer les problèmes liés à la latence. Après cette description, présentez la conception logicielle. Pour celle-ci, il est fortement recommandé de former une première partie indépendante de toute librairie graphique, puis de présenter d'autres parties qui l'implémente pour une librairie particulière. Enfin, toutes les classes de la première partie doivent avoir pour unique dépendance les classes d'état de la section précédente.

3.1 Stratégie de rendu d'un état

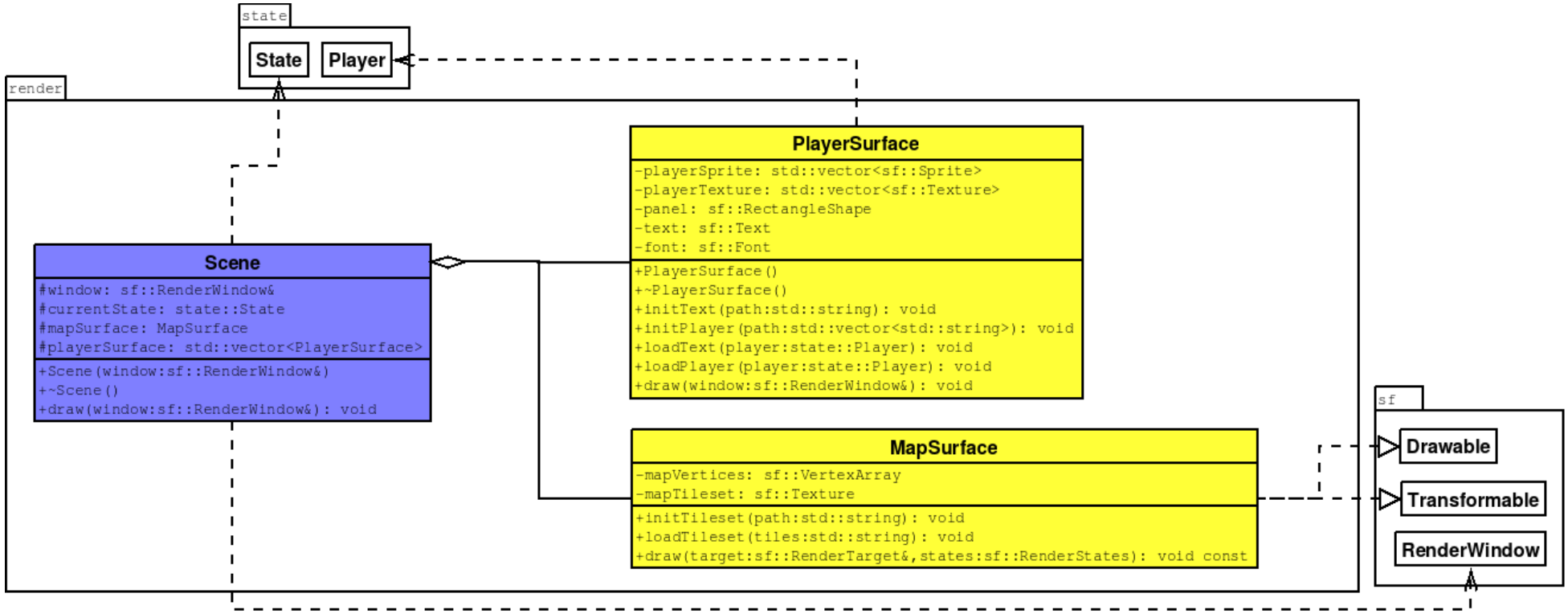
3.2 Conception logiciel

3.3 Conception logiciel : extension pour les animations

3.4 Ressources

3.5 Exemple de rendu

Illustration 2: Diagramme de classes pour le rendu



4 Règles de changement d'états et moteur de jeu

Dans cette section, il faut présenter les événements qui peuvent faire passer d'un état à un autre. Il faut également décrire les aspects lié au temps, comme la chronologie des événements et les aspects de synchronisation. Une fois ceci présenté, on propose une conception logiciel pour pouvoir mettre en œuvre ces règles, autrement dit le moteur de jeu.

4.1 Horloge globale

4.2 Changements extérieurs

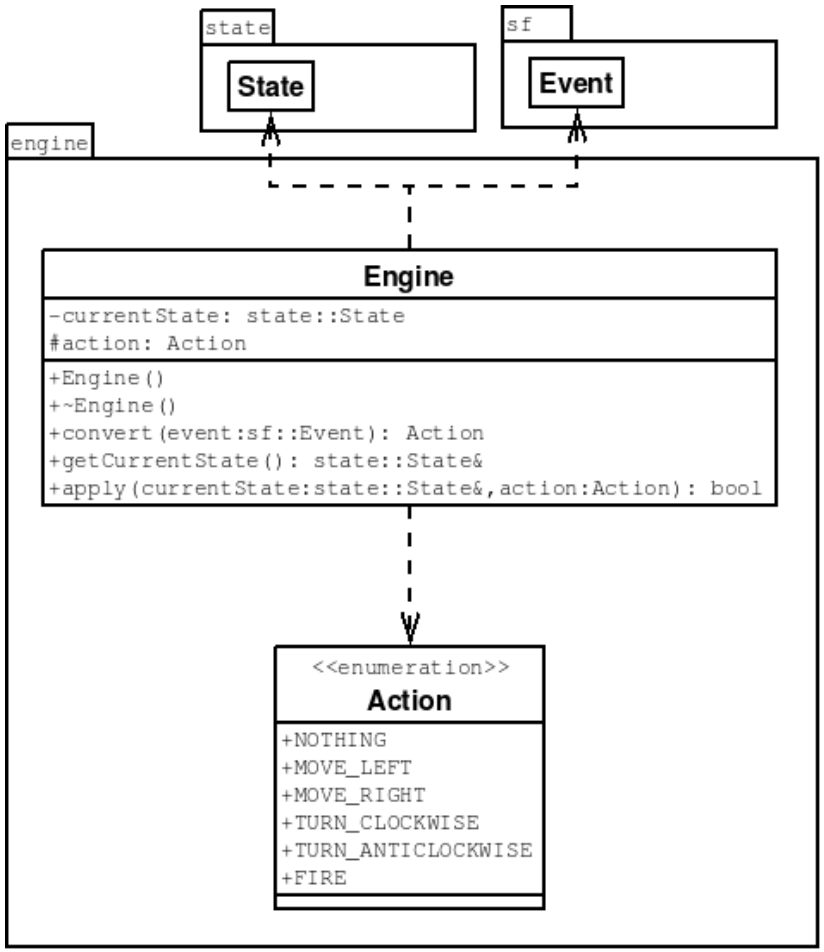
4.3 Changements autonomes

4.4 Conception logiciel

4.5 Conception logiciel : extension pour l'IA

4.6 Conception logiciel : extension pour la parallélisation

Illustration 3: Diagrammes des classes pour le moteur de jeu



5 Intelligence Artificielle

Cette section est dédiée aux stratégies et outils développés pour créer un joueur artificiel. Ce robot doit utiliser les mêmes commandes qu'un joueur humain, ie utiliser les mêmes actions/ordres que ceux produit par le clavier ou la souris. Le robot ne doit pas avoir accès à plus information qu'un joueur humain. Comme pour les autres sections, commencez par présenter la stratégie, puis la conception logicielle.

5.1 Stratégies

5.1.1 Intelligence minimale

5.1.2 Intelligence basée sur des heuristiques

5.1.3 Intelligence basée sur les arbres de recherche

5.2 Conception logiciel

5.3 Conception logiciel : extension pour l'IA composée

5.4 Conception logiciel : extension pour IA avancée

5.5 Conception logiciel : extension pour la parallélisation

6 Modularisation

Cette section se concentre sur la répartition des différents modules du jeu dans différents processus. Deux niveaux doivent être considérés. Le premier est la répartition des modules sur différents threads. Notons bien que ce qui est attendu est un parallélisation maximale des traitements: il faut bien démontrer que l'intersection des processus communs ou bloquant est minimale. Le deuxième niveau est la répartition des modules sur différentes machines, via une interface réseau. Dans tous les cas, motivez vos choix, et indiquez également les latences qui en résulte.

6.1 Organisation des modules

6.1.1 Répartition sur différents threads

6.1.2 Répartition sur différentes machines

6.2 Conception logiciel

6.3 Conception logiciel : extension réseau

6.4 Conception logiciel : client Android

Illustration 4: Diagramme de classes pour la modularisation