Projet Logiciel Transversal

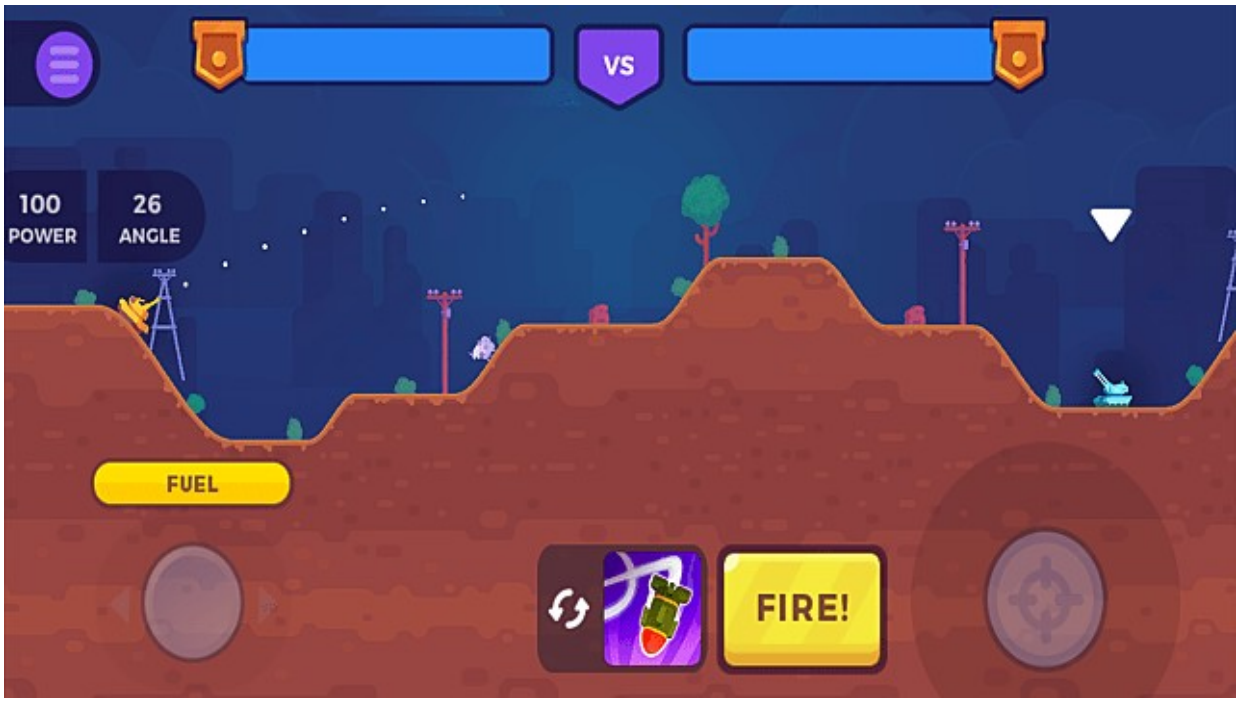
Eddy LOBJOIS – Shuman LIN – Siaka OUATTARA

Table des matières

1 Objectif....3
1.1 Présentation générale....3
1.2 Règles du jeu....3
1.3 Conception Logiciel....3
2 Description et conception des états....4
2.1 Description des états....4
2.2 Conception logiciel....4
2.3 Conception logiciel : extension pour le rendu....4
2.4 Conception logiciel : extension pour le moteur de jeu....4
2.5 Ressources....4
3 Rendu : Stratégie et Conception....6
3.1 Stratégie de rendu d'un état....6
3.2 Conception logiciel....6
3.3 Conception logiciel : extension pour les animations....6
3.4 Ressources....6
3.5 Exemple de rendu....6
4 Règles de changement d'états et moteur de jeu....8
4.1 Horloge globale....8
4.2 Changements extérieurs....8
4.3 Changements autonomes....8
4.4 Conception logiciel....8
4.5 Conception logiciel : extension pour l'IA....8
4.6 Conception logiciel : extension pour la parallélisation....8
5 Intelligence Artificielle....10
5.1 Stratégies....10
5.1.1 Intelligence minimale....10
5.1.2 Intelligence basée sur des heuristiques....10
5.1.3 Intelligence basée sur les arbres de recherche....10
5.2 Conception logiciel....10
5.3 Conception logiciel : extension pour l'IA composée....10
5.4 Conception logiciel : extension pour IA avancée....10
5.5 Conception logiciel : extension pour la parallélisation....10
6 Modularisation....11
6.1 Organisation des modules....11
6.1.1 Répartition sur différents threads....11
6.1.2 Répartition sur différentes machines....11
6.2 Conception logiciel....11
6.3 Conception logiciel : extension réseau....11
6.4 Conception logiciel : client Android....11

1 Objectif

1.1 Présentation générale

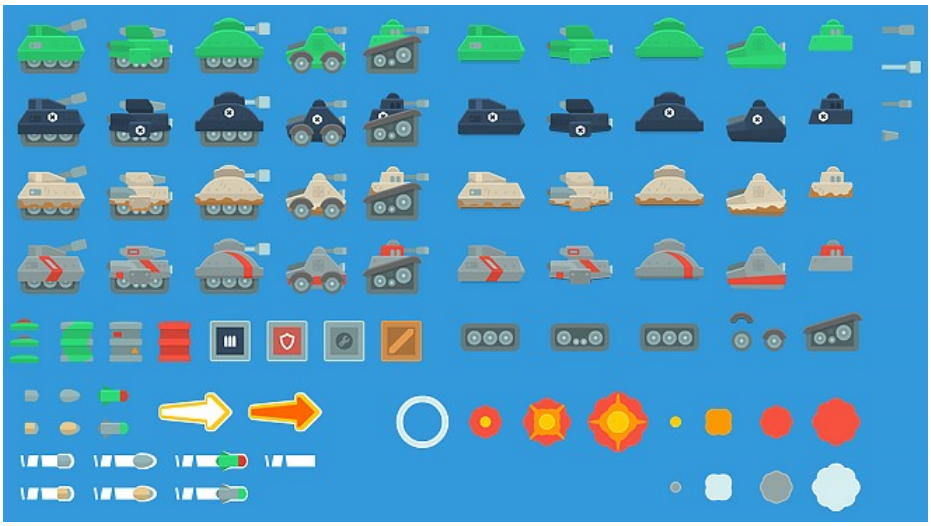


Le projet de ce semestre va s'appuyer sur le jeu Tank Stars. Il s'agit d'un jeu avec des mécanismes de gameplay similaires à Worms, mais où les vers de terre ont été remplacés par des tanks.
Dans ce jeu, deux tanks s'affrontent tour par tour dans une bataille dont le but est de détruire le tank adverse.

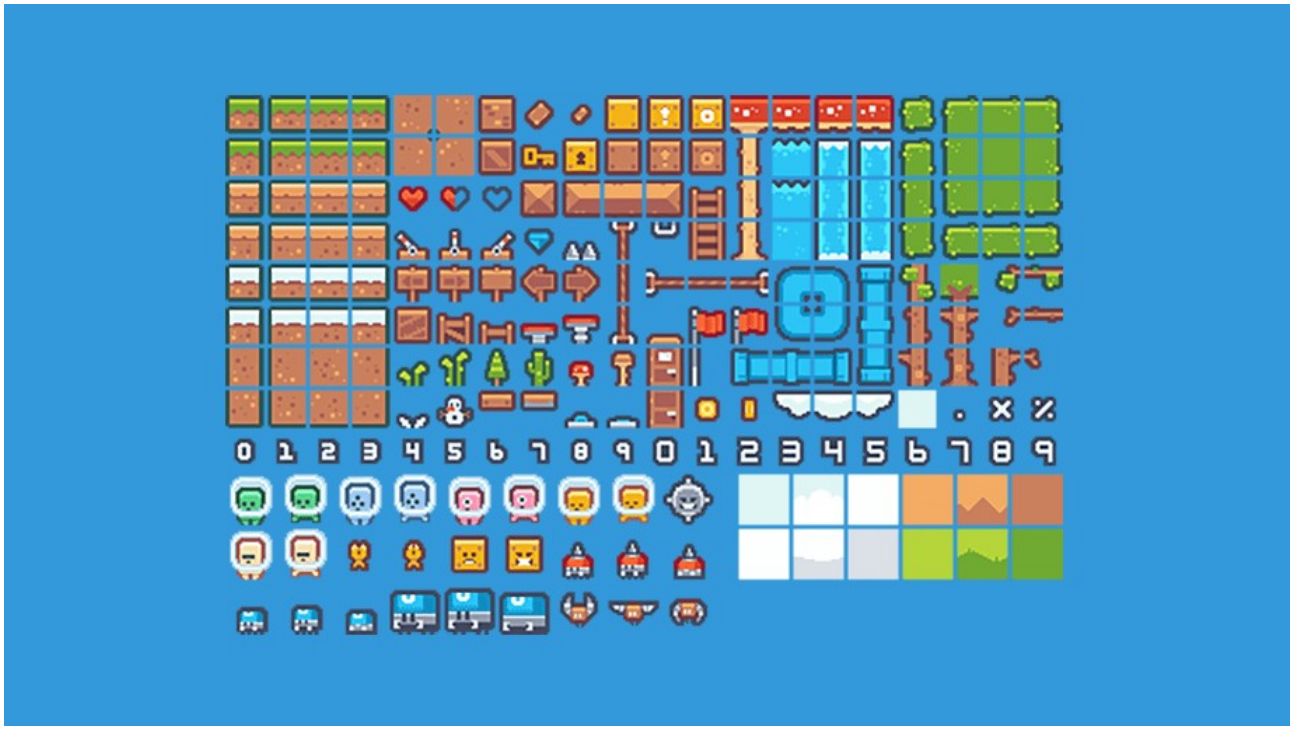
1.2 Règles du jeu

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



Bibliothèque de sprites utilisés (source: kenney.nl)



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

2 Description et conception des états

2.1 Description des états

Quatre états de jeu sont possibles:
- **GAMEOVER**: le jeu est fini.
- **MOVING**: le joueur peut se déplacer, ajuster la trajectoire angulaire de canon et modifier la puissance de son tir.
- **SHOOTING**: le joueur ne peut plus effectuer d'action, son tir est lancé en suivant une trajectoire oblique.
- **AWAITING** : le joueur est en attente du tour joué par le joueur adverse.

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 comprenant chacun un tank, un canon et un tir. Ces derniers héritent de la classe **Element** où les positions x, y et l'angle sont présents.

Pour assurer l'interdépendance entre ces trois éléments, nous avons introduit un pattern basé sur une chaîne de responsabilité tel que **tank → turret → bullet**. Le principe est que chaque modification sur un élément d'une classe mère entraîne automatiquement une modification sur toutes les classes fille de cet élément.
Ainsi, lorsqu'un tank se déplace, la tourelle et le tir se déplacent simultanément.
Lorsque la tourelle tourne, le tir tourne simultanément.
- Un attribut **turnID** qui détermine l'identifiant du joueur qui est autorisé à jouer le tour à l'instant présent.
- Un attribut **playerID** qui détermine l'identifiant du joueur.

2.3 Conception logiciel: extension pour le rendu

La classe **Scene** est constituée d'une fenêtre de rendu **window** comprenant:
- Une classe **MapSurface** qui s'occupe du chargement et de l'affichage graphique des tiles comprenant la map.
- Une classe **PlayerSurface** qui s'occupe du chargement et de l'affichage graphique des éléments de chaque joueur et des informations textuelles associées. Cette classe est déclarée autant de fois qu'il y a de joueurs (deux joueurs dans ce jeu).

Le rendu graphique de ces deux classes est geré par une méthode abstraite **draw**.

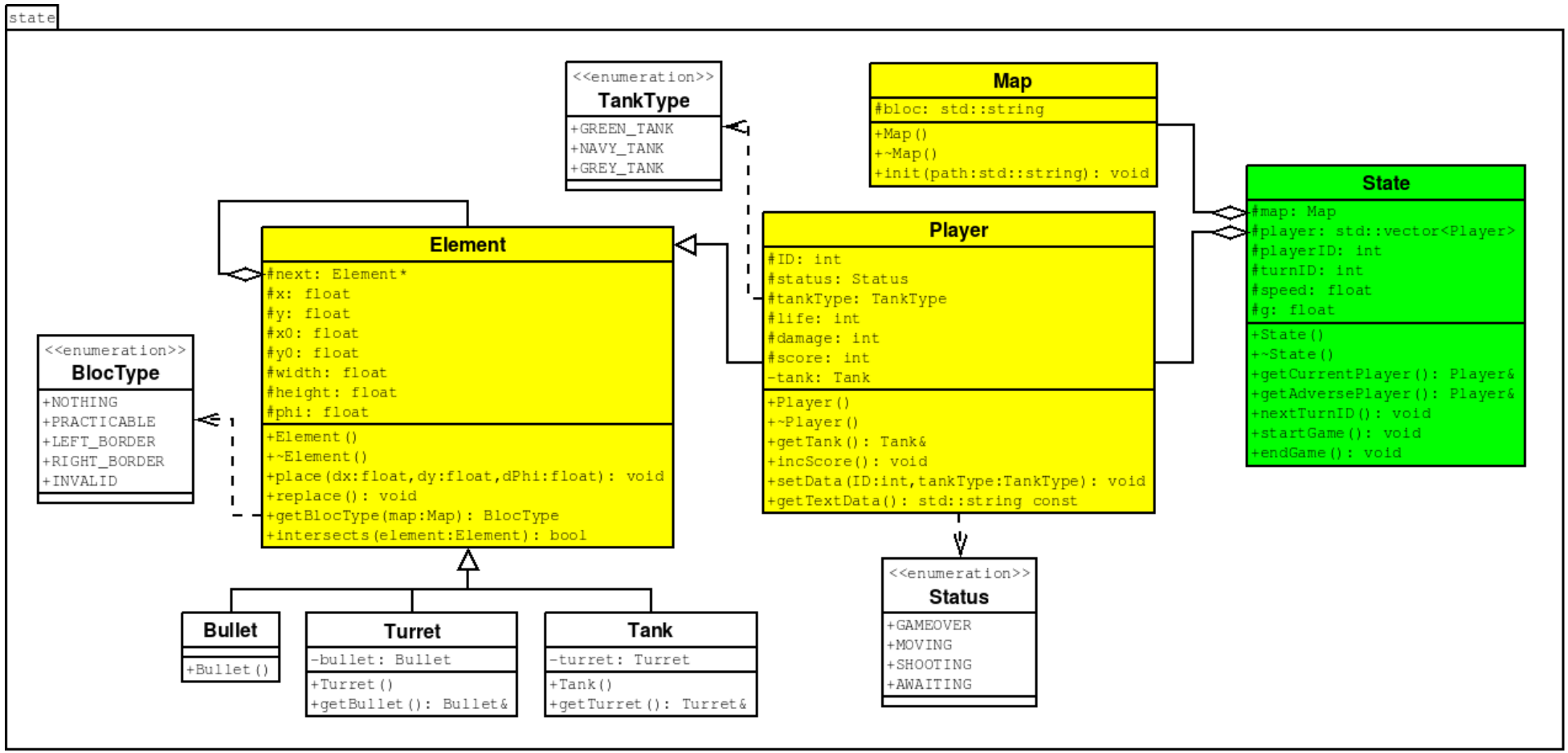
2.4 Conception logiciel: extension pour le moteur de jeu

Le moteur de jeu est utilisé pour gérer la mise à jour des états en fonction de chaque nouvelle action du jeu.
Il est composé d'une classe abstraite **Command** appelant la classe **KeyboardCommand** chargé de convertir un appui sur une touche du clavier en une action de jeu dans l'hypothèse où le mouvement est valide. La méthode **run** sera utilisée ici.

Les six actions valides sont:
- **NOTHING** (ne fait rien)
- **MOVE_LEFT** (déplacement d’un tank à gauche)
- **MOVE_RIGHT** (déplacement d’un tank à droite)
- **TURN_ANTICLOCKWISE** (rotation du canon dans le sens antihoraire)
- **TURN_CLOCKWISE** (rotation du canon dans le sens horaire)
- **FIRE** (tir du projectile).

2.5 Ressources

Illustration 1: Diagramme des classes d'état



3 Rendu: Stratégie et Conception

3.1 Stratégie de rendu d'un état

Pour le rendu graphique du jeu, nous avons utilisé la librairie SFML qui comprend toutes les fonctions nécessaires pour l'affichage des formes, des textes et des sprites.

3.2 Conception logiciel

Le rendu graphique du jeu est partagé entre les deux classes présentes dans **Scene**:
- **MapSurface** qui utilise les informations contenu dans **State::Map**. Tous les éléments graphiques de la map sont contenues dans une même tileset (voir l'onglet ressources 3.4). La méthode **loadTileset()** se charge d'afficher le bon segment de la tileset en parcourant la surface de l'écran en fonction le l'indice n du vecteur logique **State::bloc** associé.
- **PlayerSurface** qui utilise les informations contenu dans **State::Player**. Cette classe est instanciée autant de fois qu'il y a de joueurs (ici deux). Elle affiche le tank, sa tourelle et son tir.
Un attribut textuel **text** permet de visualiser les informations de chaque joueur (état, position, angle...)

L'affichage de chaque élément graphique est effectué par la méthode **draw()**.

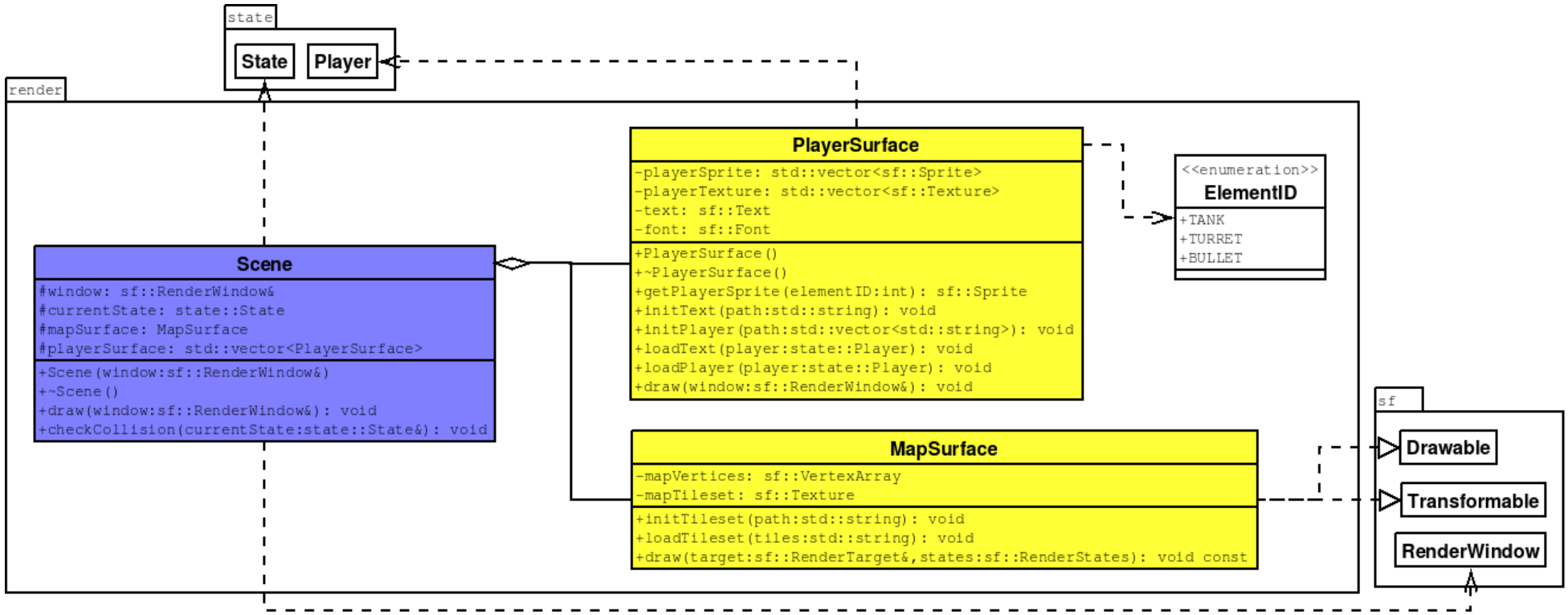
3.3 Conception logiciel: extension pour les animations

3.4 Ressources

Tileset utilisé pour le rendu de la map

3.5 Exemple de rendu

Illustration 2: Diagramme de classes pour le rendu



4 Règles de changement d'états et moteur de jeu

4.1 Horloge globale

- Si l'état présent est **GAMEOVER** et l'action retournée est **START_GAME**, l'état futur passe à **MOVING**.

- Si l'état présent est **MOVING**, les cinq actions prédéfinies (**MOVE_LEFT**, **MOVE_RIGHT, TURN_ANTICLOCKWISE**, **TURN_CLOCKWISE** et **FIRE**) sont rendues possibles.
Dans ce cas l'action **FIRE** fait passer l'état futur à **SHOOTING**.

- Si l'état présent est **SHOOTING**, aucune action n'est possible et l'état passe à **GAMEOVER** si le tir touche un tank et que ce dernier à un nombre points de vie ≤ 0. Si le tir n'est pas fatal ou qu'il touche un élément du décor, l'état futur repasse à **MOVING**.

4.2 Changements extérieurs

L'angle du canon devra être obligatoirement compris entre -180° et 0°.
Parallèlement, le tank ne pourra plus se déplacer que dans le sens inverse si le sprite associé est en collision avec une bordure **LEFT_BORDER** ou **RIGHT_BORDER**.
Dans le cas contraire, l'action sera invalidée par le biais de la valeur de retour **NOTHING**.

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

4.3 Changements autonomes

4.4 Conception logiciel

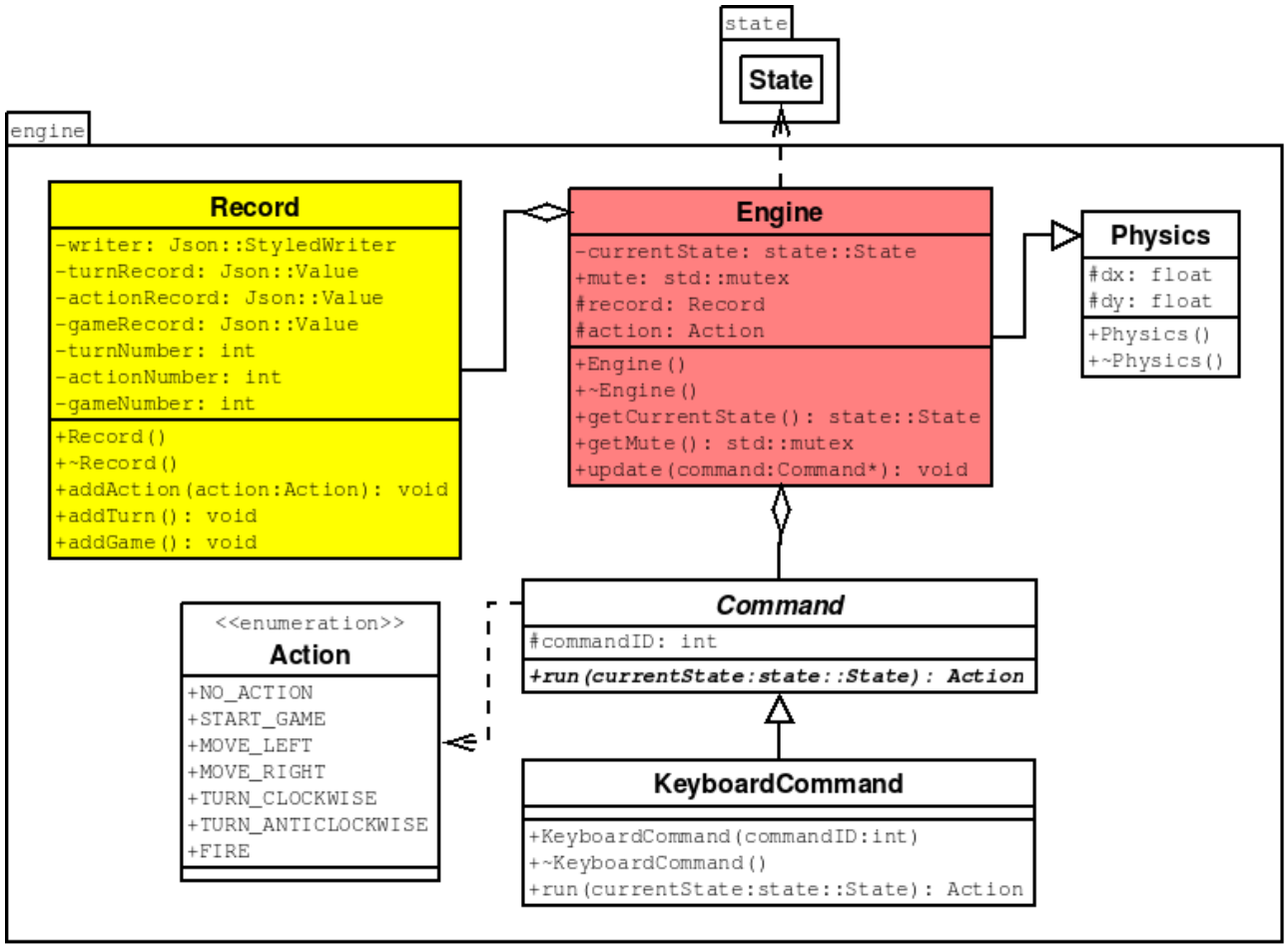
Pour la mise à jour de l'affichage, l'état du moteur de jeu va être recopié dans l'état du rendu graphique à chaque appel de la fonction **display()**.

4.5 Conception logiciel : extension pour l'IA

Pour simplifier l'appel de chaque IA, celles-ci sont organisées selon le pattern strategy ou les classes **DumbAI**, **HeuristicAI** et **AdvancedAI** héritent toutes de la classe abstraite **AI**, qui hérite elle-même de la classe **Command**.

4.6 Conception logiciel : extension pour la parallélisation

Illustration 3: Diagrammes des classes pour le moteur de jeu



5 Intelligence Artificielle

5.1 Stratégies

5.1.1 Intelligence minimale

L'IA aléatoire consiste à effectuer dans un premier un nombre **maxIteration** de fois une action définie aléatoirement parmi **MOVE_LEFT**, **MOVE_RIGHT, TURN_ANTICLOCKWISE** ou **TURN_CLOCKWISE**.
Ensuite, l'IA termine son tour en effectuant l'action **FIRE**.

5.1.2 Intelligence basée sur des heuristiques

L'IA heuristique consiste à obtenir la distance horizontale entre les deux tanks, puis à évaluer l'angle de portée optimal à partir duquel le tir va toucher la cible adverse à chaque tour.

L'équation mathématique est la suivante:

$\theta = 0.5*\arcsin(g*d/v^2)$

Où g représente la constante gravitationnelle du jeu et v représente la vitesse du projectile.

Pour éviter que le tir ne réussisse à toucher la cible adverse à 100% des essais, nous avons introduit une variable **delta** qui incrémente ou décrémente aléatoirement l'angle de tir optimal d'une valeur donnée.

5.1.3 Intelligence basée sur une descente de gradient

L'IA avancée consiste à implémenter un algorithme de descente de gradient qui affine la trajectoire du tir en fonctions du niveau d'entraînement de l'IA.
A chaque fin d'un tour joué, l'IA va comparer la position du tir avec la position du tank adverse, puis corriger un coefficient **w** selon l'équation suivante:

$w_{n+1} = w_n + k * (x_{tir} - x_{tank\ adverse})$

Où **k** représente un coefficient d'apprentissage qui doit être judicieusement choisi selon un compromis entre vitesse d'apprentissage et stabilité.
Si **k** est correctement choisi, la valeur de **w** doit converger vers $\mathbf{g/v^2}$.

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

6.1 Organisation des modules

6.1.1 Répartition sur différents threads

Le fonctionnement du jeu est divisé en trois threads:
- Un thread gérant le rendu graphique du jeu.
- Deux threads séparés pour gérer indépendamment le fonctionnement de chaque joueur.

Pour résoudre les problèmes liés à l'accès concurrentiel aux ressources mémoire et plus particulièrement au moteur de jeu, nous avons introduit un mutex qui permet de verrouiller un thread au du début de son exécution et de le déverrouiller à la fin de son exécution.

L'intérêt technique de séparer le thread d'affichage des threads de rendu est de pouvoir effectuer par la suite des simulations testant le pourcentage de parties gagnées d'une IA sans affichage graphique, ce qui est possible beaucoup plus rapidement qu'avec l'affichage graphique (le gain est de l'ordre de x100).

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