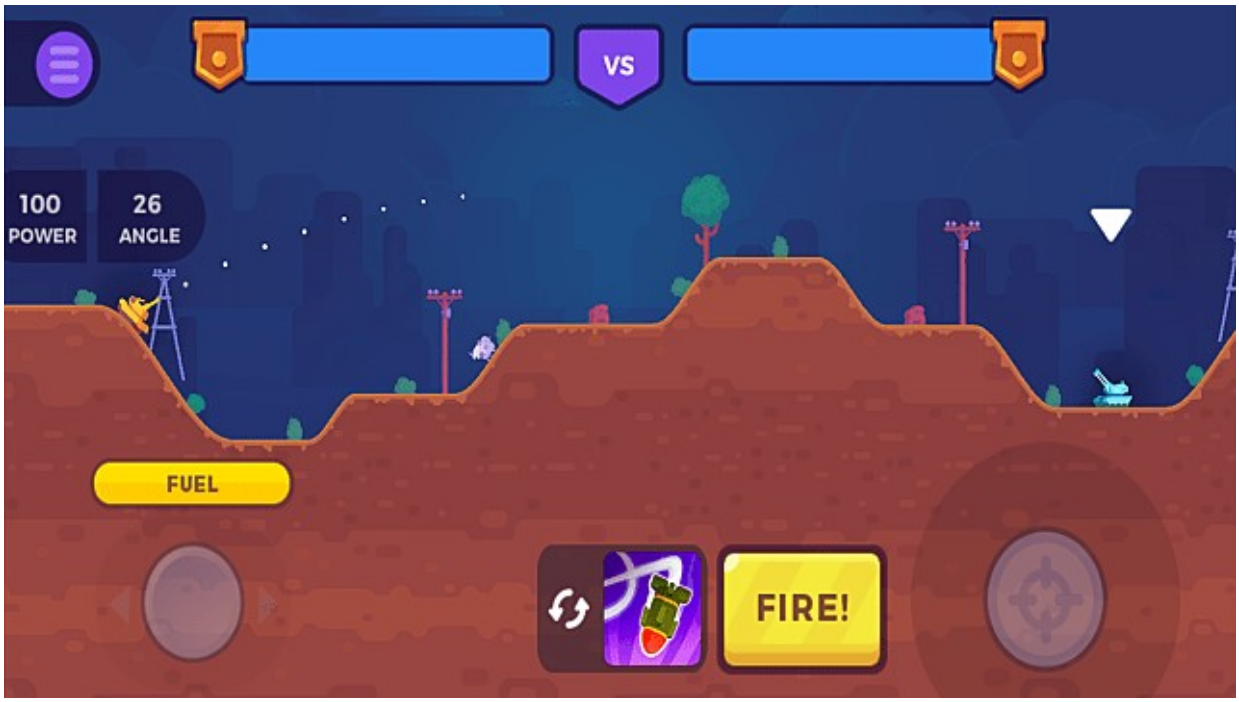
Projet Logiciel Transversal

Eddy LOBJOIS – Shuman LIN – Siaka OUATTARA

Table des matières

1 Objectif

1.1 Présentation générale

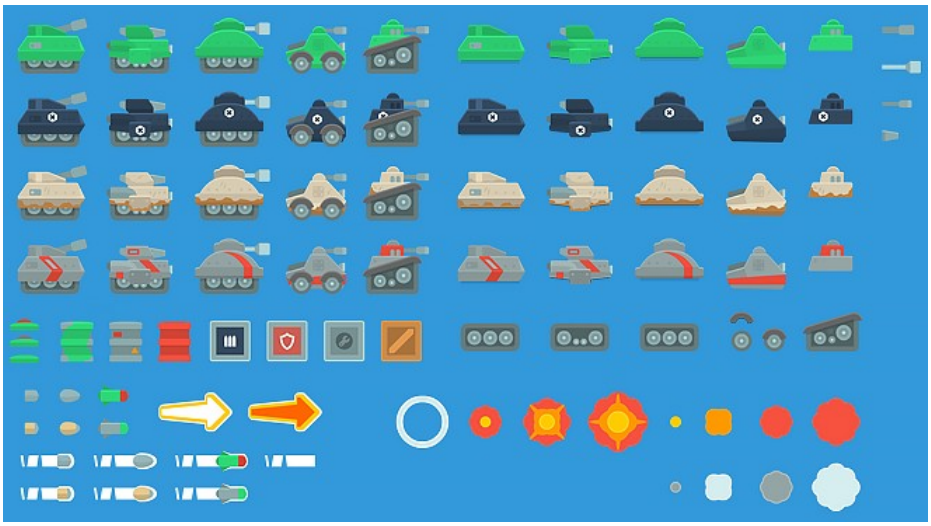


Le projet de ce semestre va s'appuyer sur le jeu Tank Stars. Il s'agit d'un jeu avec des mécanismes de gameplay similaires à Worms, mais où les vers de terre ont été remplacés par des tanks.
Dans ce jeu, deux tanks s'affrontent tour par tour dans une bataille dont le but est de détruire le tank adverse.

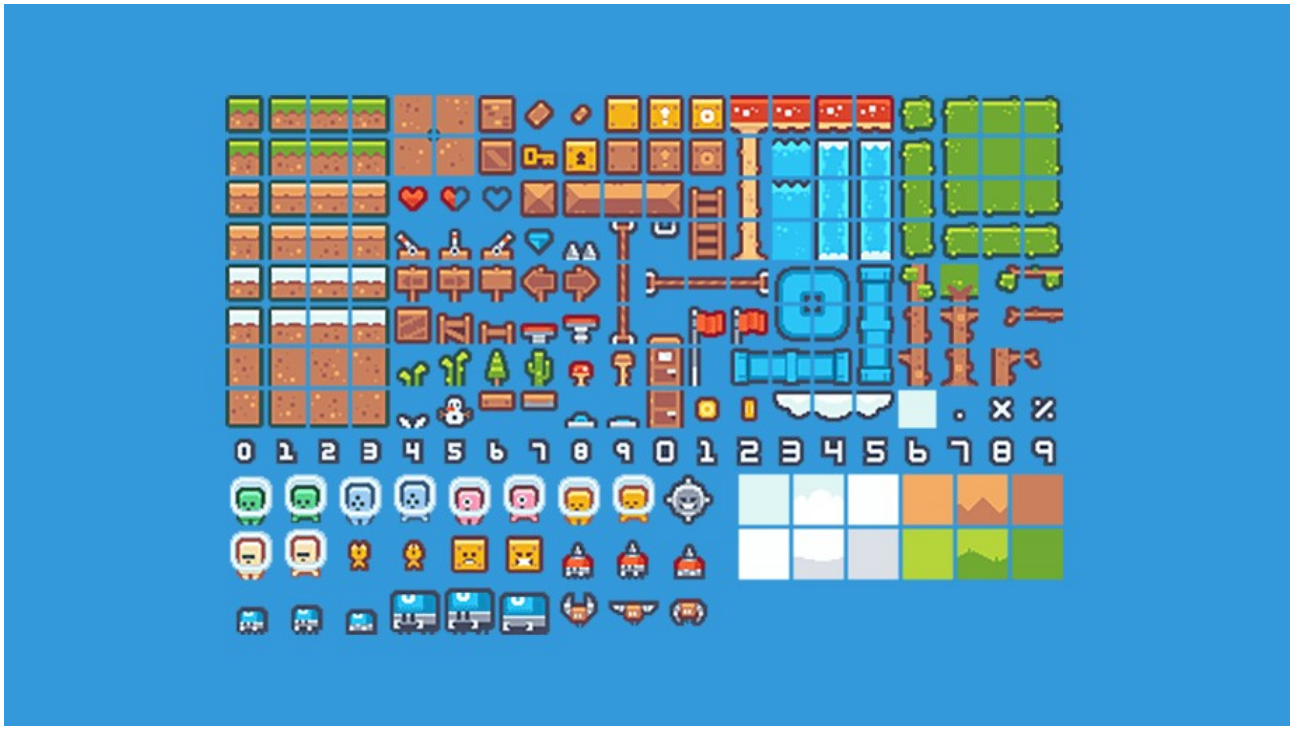
1.2 Règles du jeu

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

Trois 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.

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 le tank, le canon et le tir. Ces derniers héritent de la classe **Element** où les positions x, y et l'angle sont présents
- 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 de deux classes:
- Une classe **Command** 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 **convert** sera utilisée ici.

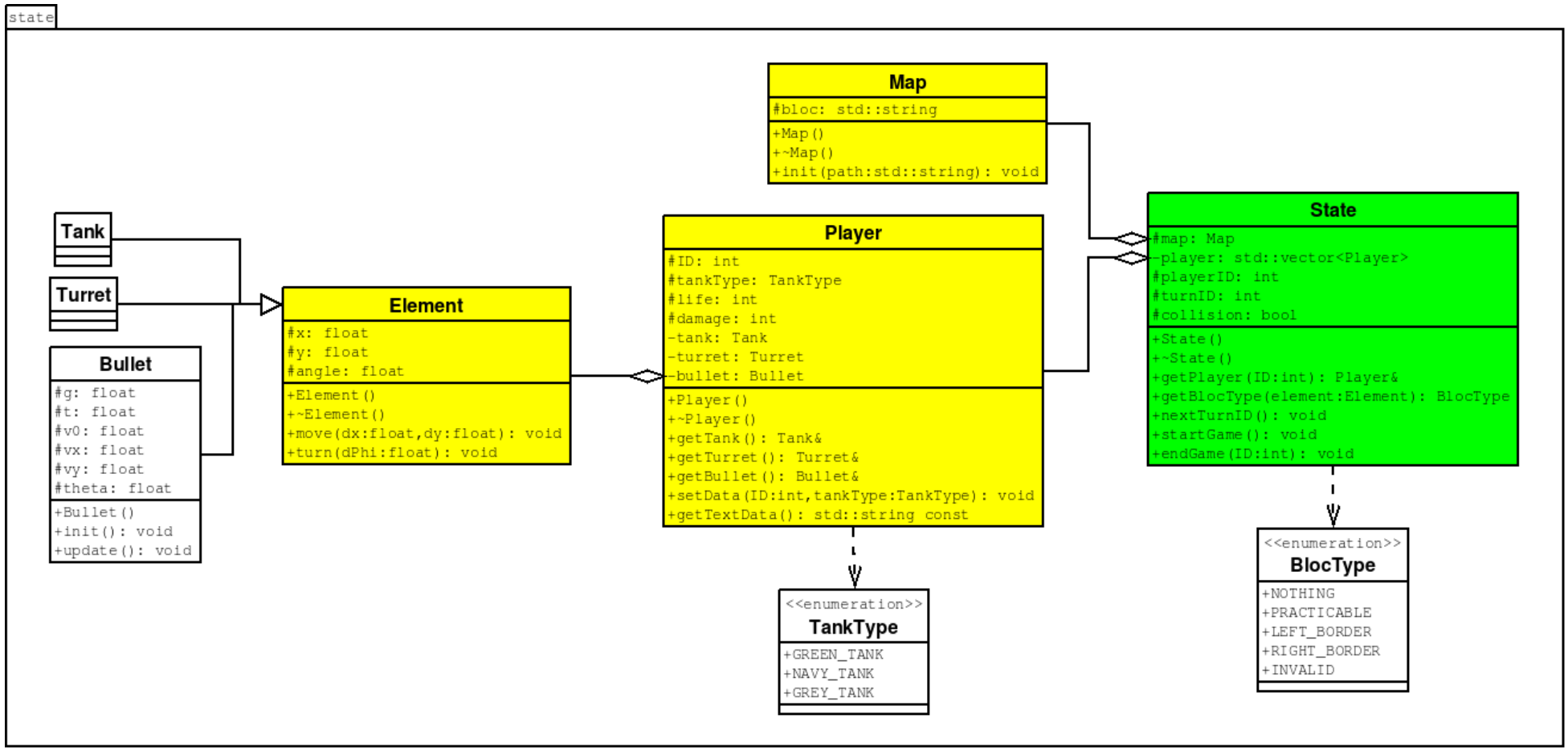
Les cinq actions valides sont:
- **MOVE_LEFT** (déplacement d'un tank à gauche)
- **MOVE_RIGHT** (déplacement d'un tank à droite)
- **TURN_ANTICLOCKWISE** (rotation du canon dans le sens antihoraire)
- **TURN_CLOCKWISE** (rotation du canon dans le sens horaire)
- **FIRE** (tir du projectile).

Trois états de jeu sont possibles:
- **GAMEOVER**: le jeu est fini.
- **MOVING**: 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.

2.5 Ressources

Tileset utilisé pour le rendu de la map

Illustration 1: Diagramme des classes d'état



3 Rendu : Stratégie et Conception

3.1 Stratégie de rendu d'un état

- Si l’état présent est **GAMEOVER** et l’action retournée est **FIRE**, 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**.

3.2 Conception logiciel

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**.

3.3 Conception logiciel : extension pour les animations

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

3.4 Ressources

3.5 Exemple de rendu

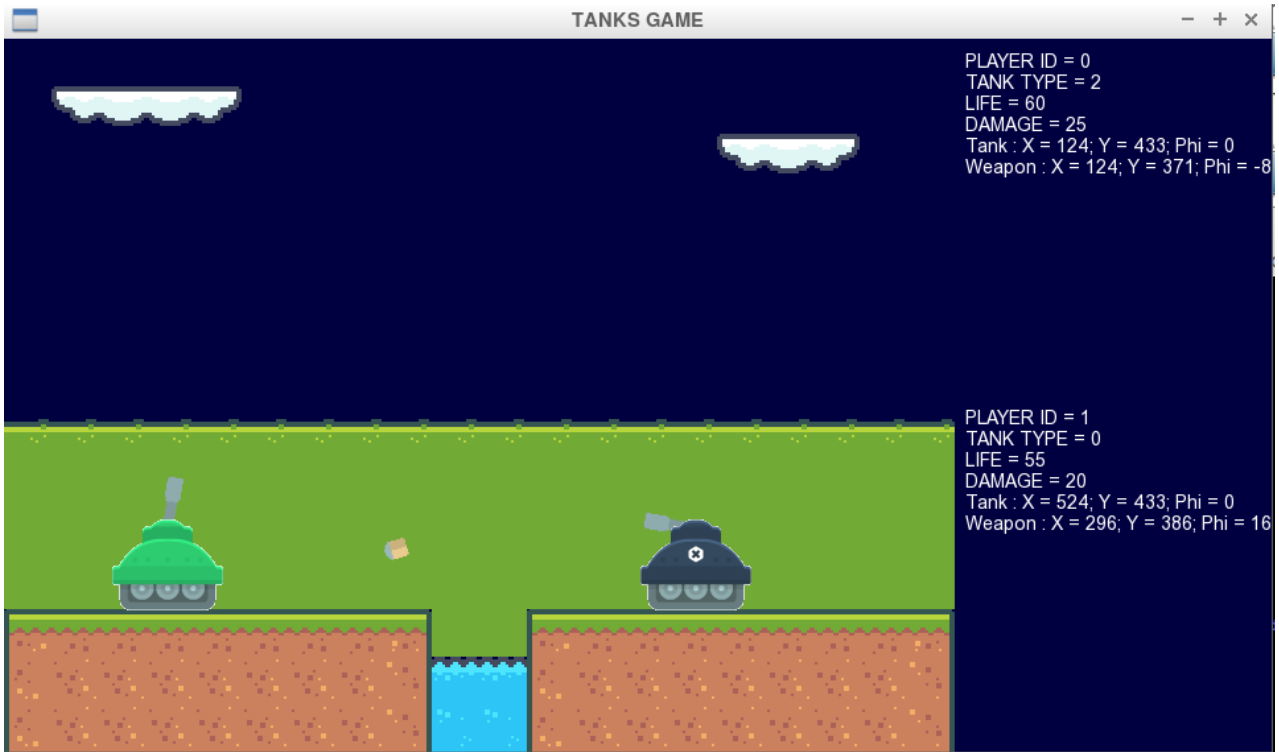
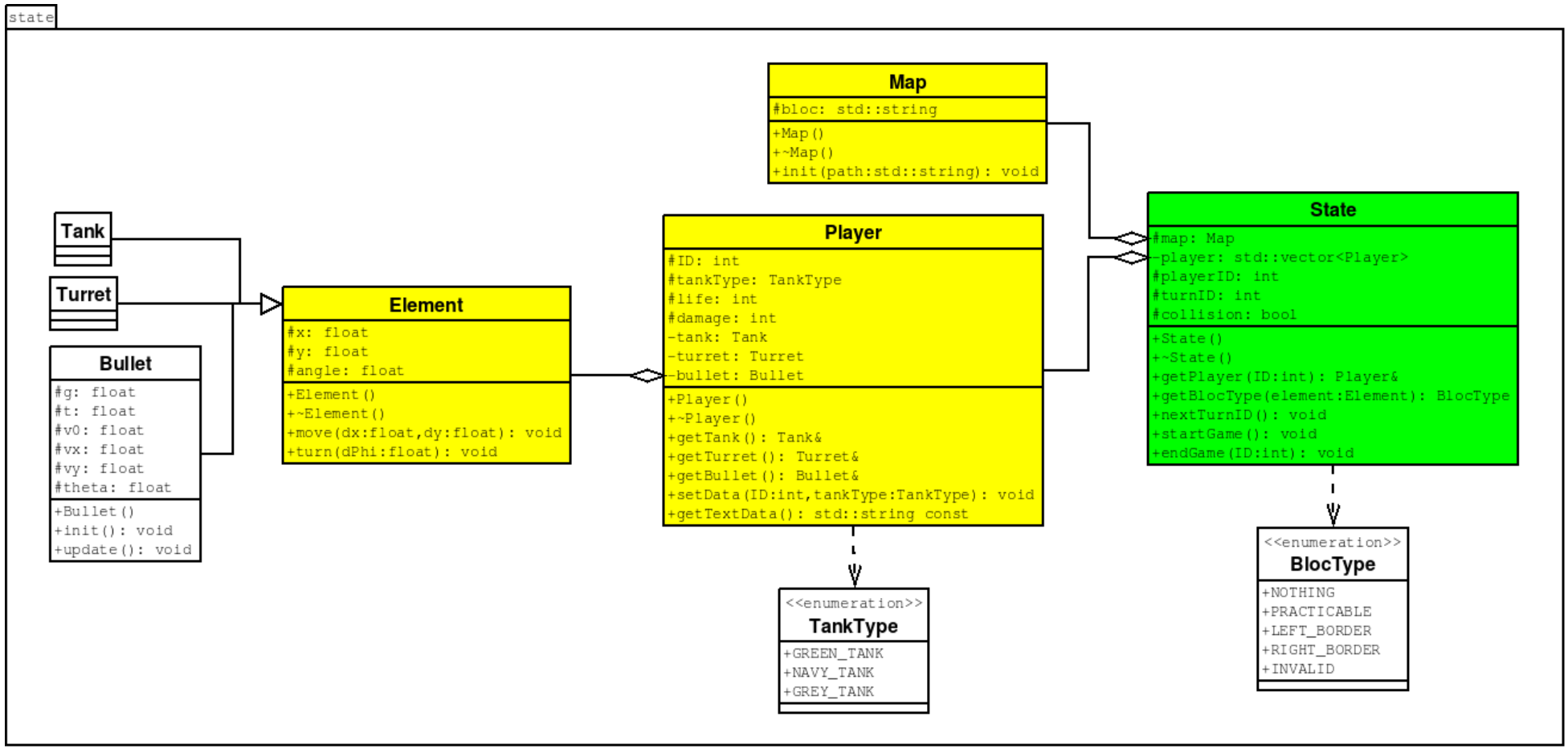


Illustration 2: Diagramme de classes pour le rendu



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

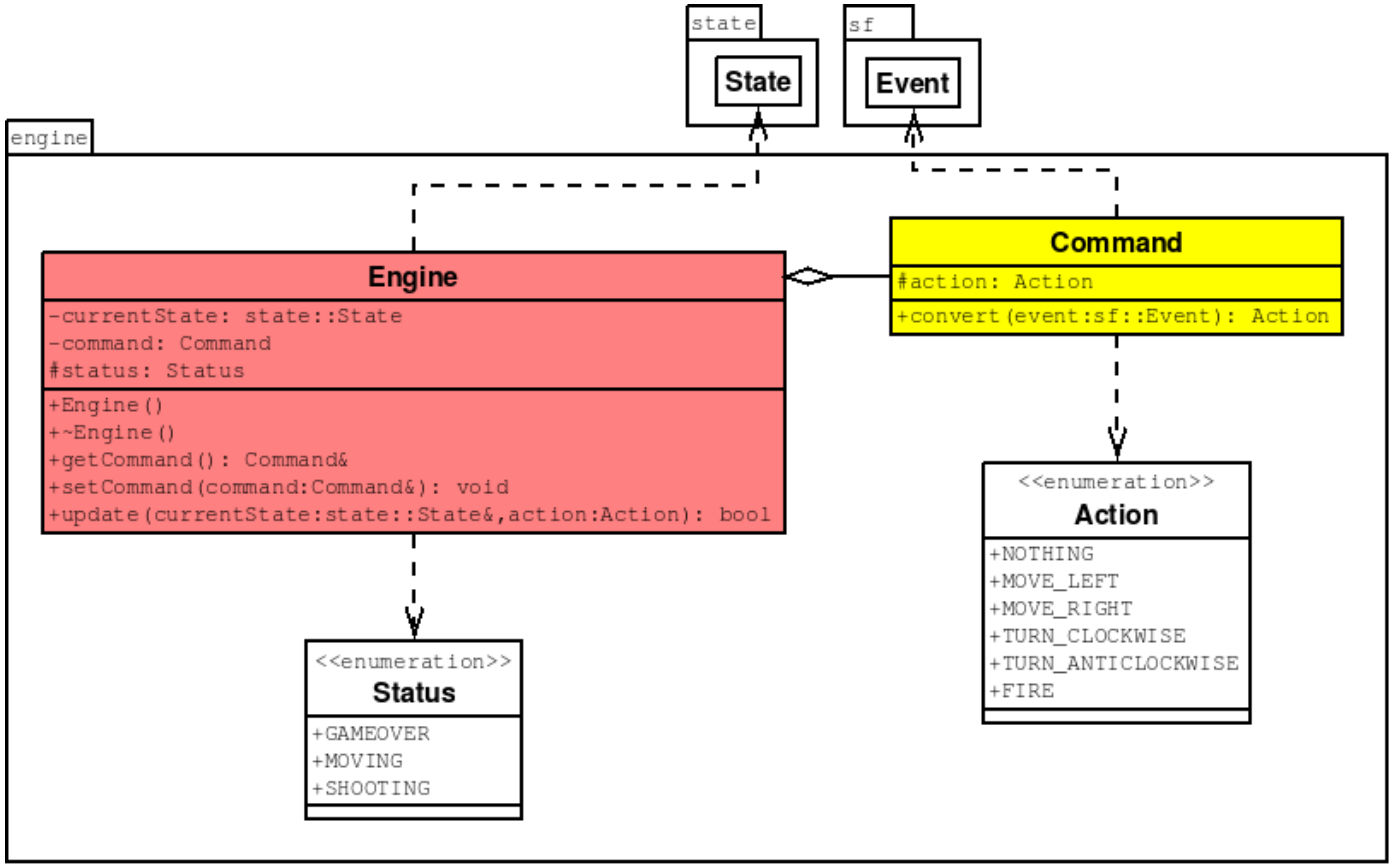
4.3 Changements autonomes

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

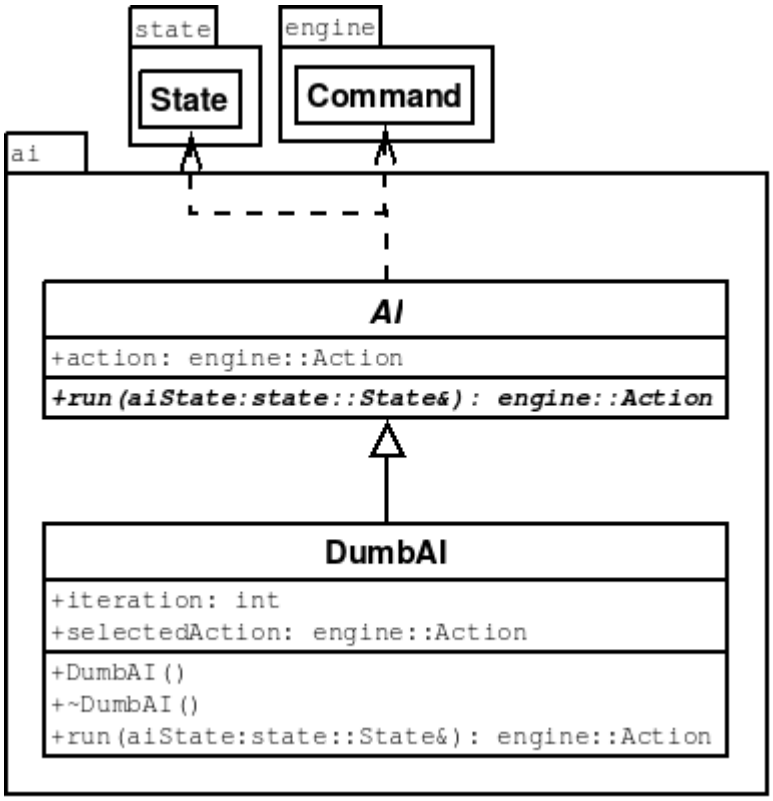
5.1.3 Intelligence basée sur les arbres de recherche

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


state
State
engine
Command
ai
AI
+action: engine::Action
+run(aiState:state::State&): engine::Action
DumbAI
+iteration: int
+selectedAction: engine::Action
+DumbAI()
+~DumbAI()
+run(aiState:state::State&): engine::Action

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