ENSEA
Ecole Nationale
Supérieure
de l'Electronique
et de ses Applications

Projet Logiciel Transversal

Alexis LOISON – Eric TAN

Option Informatique et Systèmes

Capture d'écran du jeu

SOMMAIRE

1 Objectif

1.1 Présentation générale

Présenter ici une description générale du projet. On peut s'appuyer sur des schémas ou croquis pour illustrer cette présentation. Éventuellement, proposer des projets existants et/ou captures d'écrans permettant de rapidement comprendre la nature du projet.

Le jeu proposé est basé sur « Advance Wars », un jeu de guerre militaire en 2 dimensions avec la possibilité d'avoir plusieurs joueurs qui puissent y jouer. Nous travaillerons avec 2 joueurs ou bien 1 joueur contre 1 IA.

Capture d'écran du jeu

1.2 Règles du jeu

Les joueurs doivent capturer toutes les infrastructures ennemies ou uniquement le quartier général.
Si le joueur perd toutes ses infrastructures ou son quartier général il perd la partie.
Chaque joueur peut créer des unités à partir de bâtiments de production qui seront construits grâce à des ressources récoltées/perçues en jeu.
Chaque unité a des points de vie, des avantages et des faiblesses face à d'autres unités. Elle pourra des points de vie en retournant à l'usine(terrestres) ou à l'aéroport(aériens).

1.3 Conception Logiciel

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

Package state : il s'agit du package contenant toutes les informations pour représenter un état de notre jeu.
Package render : celui-ci permet d'afficher un état de notre jeu en version graphique.

2 Description et conception des états

2.1 Description des états

L'ensemble des états fixes et mobiles sont composés de :

- Coordonnées (x,y) dans la grille de chaque entité
- Identifiant du type d'élément : la nature de l'élément et sa couleur

2.1.1 Etat éléments fixes

Notre carte sera composée de « cases ». La taille est définie par défaut. Chaque case ne pourra accueillir qu'un maximum de 10 unités du même élément à la fois.

- **Cases « Obstacles »**. Celles-ci sont infranchissables par certains éléments mobiles ou franchissables avec pénalités.
  - ➔ « Rivière » : franchissable uniquement par les éléments mobiles « aériens »
  - ➔ « Forêt » : franchissable par les éléments mobiles « aériens » et les éléments « infanteries » avec pénalité

- **Cases « Franchissables ».** Toutes les unités mobiles peuvent franchir ces éléments.
  - ➔ « Vide » : case ayant pour texture esthétique : une route ou de l'herbe
  - ➔ « Mine » : case rapportant des ressources de métal au détenteur
  - ➔ « Immeuble » : procure une diminution des dégâts reçus pour les infanteries
  - ➔ « Quartier Général » : permet aux éléments mobiles de récupérer des points de vie. La capture de cette case par un élément mobile ennemie procure la défaite de son ex-détenteur.
  - ➔ « Usine » : permet de créer des éléments « terrestres » mobiles ou leur permet récupérer des points de vie
  - ➔ « Aéroport » : permet de créer des éléments « aériens » ou leur permet récupérer des points de vie

2.1.2 Etat éléments mobiles

Les éléments mobiles possèdent :

- Un **déplacement** : vers la gauche, vers la droite, vers le haut, vers le bas
- Une **vitesse de déplacement** définissant le nombre de cases que peut parcourir un élément mobile en 1 tour de jeu
- Une **position** [x,y] définit sur le cadre
- Des **dégâts**
- Des **points** de vie : entier allant de 0 à 10
- Une **couleur** : 1 pour le rouge, 2 pour le bleu et 3 pour le gris
- Un **rang** caractérisé par un chiffre : 1 pour une promotion et 0 pour une non-promotion
- Ces éléments peuvent être « aérien » ou « terrestre ».

2.1.3 Etat général

Les propriétés suivantes sont ajoutées :
Epoque : « l'horloge » du jeu, l'heure depuis le commencement de la partie
Fréquence : nombre de fois par minute que l'état du jeu est mis à jour

2.2 Conception logiciel

Nous utilisons le logiciel Dia afin de créer notre diagramme des classes. Décrivons en deux groupes nos différentes classes.

Classes Elément : composé de *Element*, *MobileElement* et *StaticElement*, elles permettent de décrire les éléments qui composent notre jeu de type mobile ou statique. *Element* englobe les deux autres classes filles citées.

On pourra par la suite choisir si un élément est aérien ou terrestre lorsqu'il appartient hérite de la classe *MobileElement*. Dans un autre cas, si celui-ci est statique, on passera par les deux classes filles *Wall* et *Space*.

Conteneurs d'élément : les trois classes *State*, *ElementList* et *ElementGrid* permettent de contenir tous nos éléments. *ElementList* sert à stocker sous forme de liste les éléments de notre jeu. Tandis que *ElementGrid* sert à stocker notre plateau de jeu sous la forme d'une grille. Bien sûr, les deux classes contiennent également les fonctions permettant de modifier une liste d'élément pour la première et une grille pour la seconde. *State* est la classe état qui permet de rendre compte de l'instant auquel on se trouve. Ainsi, il est composé avec les deux autres classes car on a besoin d'une grille et d'une liste d'éléments.

2.3 Conception logiciel : extension pour le rendu

2.4 Conception logiciel : extension pour le moteur de jeu

2.5 Ressources

On codera notre carte en fichier texte afin de pouvoir traduire plus tard celle-ci en carte de jeu.

```
FGGGFGGGGGGGGGGGGGGGGGGGGGGG
GGGGFGGGGGGGGGGGGGGGGGGGGGGG
GRHMRGGGGRRRRRRRRRFWWGGGRSSSSSGGG
ARBBFGGGGRMSSSWSWRWWWWGGRSSSSSSGG
GRRRRRRRRRWSWWWWBRGGRRRRRRSSSSSSG
GGGGGGWRWWWWWRRRRRRRRRBWWWRSGSSSGG
GGGGGGGRWWWWWRMWGGGRSWWWWRSGSMSGG
GGGGGGWRWWWWBRWWGGSSSWWWWRAGGGGGG
GGGGGGARWWWWSSSGGWWRBWWWWRWGGGGGG
GGSMSGSRWWWWSRGGGWMRWWWWWRGGGGGGG
GGSSSGSRWWWBRRRRRRRRWWWWWRWGGGGGG
GSSSSSSRRRRRRGGRBWWWWSWRRRRRRRRRR
GGSSSSSSRGGWWWWRWSWSSSMRGGGGFBBRA
GGGSSSSSRGGGWWFRRRRRRRRRGGGGRMHRG
GGGGGSGGRRRRRRRRGSGGGGGGGGGRRRRG
GGGGGGGGGGGGGGGGGGGGGGGGGGGFGGGF
```

Pour y voir plus clairement

R/Road/Gris
G/Grass/Vert
M/Mine/Jaune
B/Building/Gris clair
H/Headquarter/ Rose
F/Factory/Orange
A/Airport/Violet
S/Sea/Bleu/Marron
W/Woods

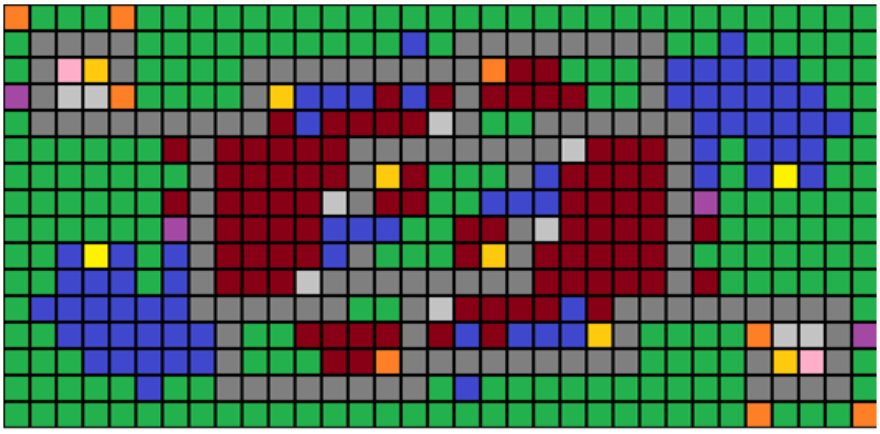
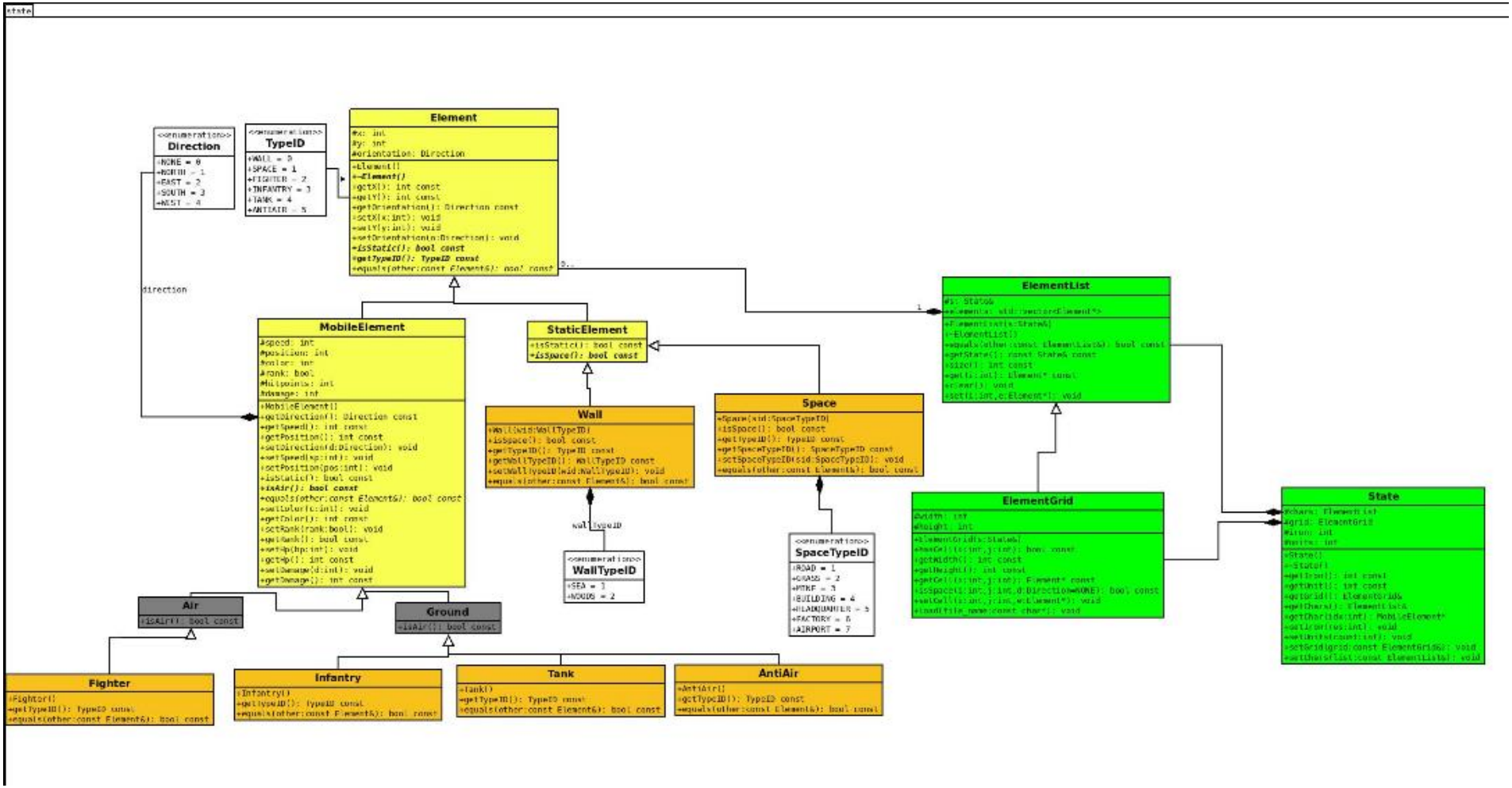


Illustration 1: Diagramme des classes d'état



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

Pour faire le rendu d'un état, nous utilisons SFML : Simple and Fast Multimedia Library. Celle-ci va nous permettre d'alléger le traitement des tuiles faites par la carte graphique de l'ordinateur.

Le principe utilise un système avec des vertex contenant plusieurs informations : une position, une couleur et une paire de coordonnées pour un jeu en 2 dimensions. Lorsque l'on groupe ces vertex ensemble (primitives), on obtient un tableau qui nous donne une forme.

Dans notre jeu, on affiche d'abord toute la carte avec les textures correspondantes aux bons tableaux de vertex. Ensuite, on peut par la suite changer les attributs des vertex qui nous intéressent pour modifier le rendu (par exemple signaler un changement d'unité/bâtiment d'une case de notre terrain).

Il faut également noter que nous aurons deux « fenêtres », une qui affiche le terrain et une autre qui montre les informations du joueur (ressources, couleur) ainsi que celle d'une unité/bâtiment qu'il aura sélectionnée.

3.2 Conception logiciel

Dans le cadre de notre jeu, nous allons utiliser le diagramme des classes ci-après.

3.3 Ressources

Voici les ressources utilisées pour notre jeu :

3.4 Exemple de rendu

Exemple de rendu du niveau de jeu :

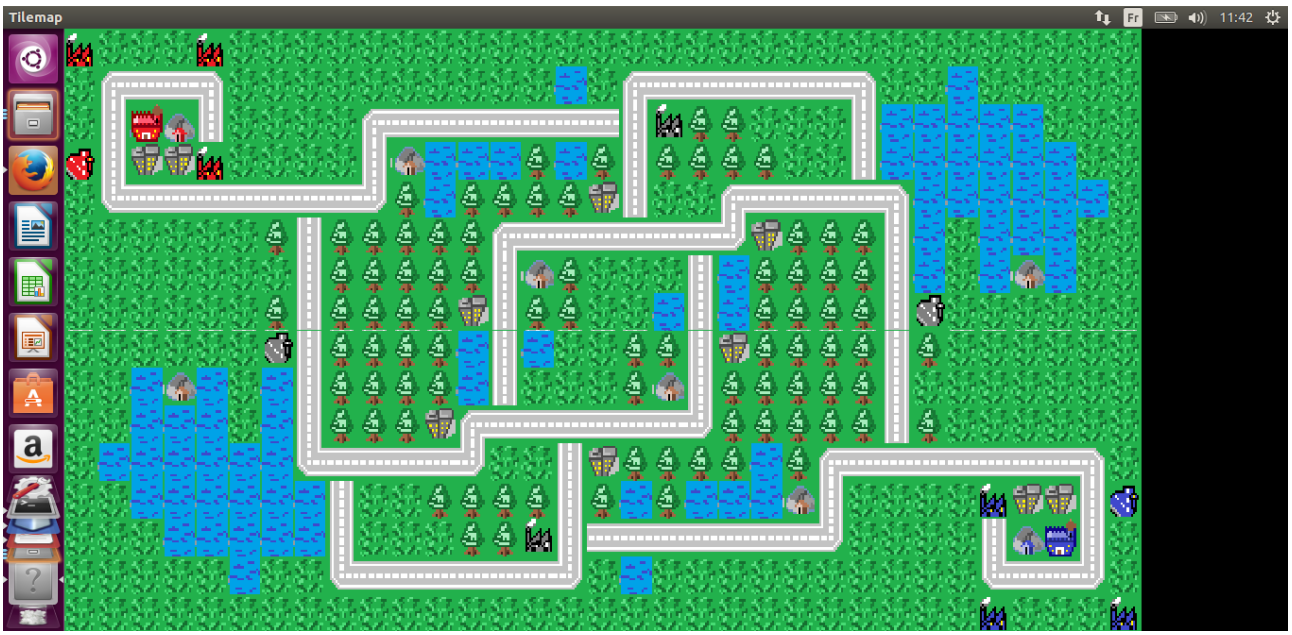

Tilemap
Fr
11:42

Illustration 2: Diagramme des classes du rendu