

Projet Logiciel Transversal

Dragons & Licornes

Kevin TOOMEY – Chloé LODOLO

Table des matières

1	Présentation générale	4
1.1	Archétype	4
1.2	Règles du jeu	4
1.3	Ressources.....	4
2	Description et conception des états	6
2.1	Description des états	6
2.1.1	Eléments fixes	6
2.1.2	Eléments variables	6
2.1.3	Etat général	6
2.2	Conception logiciel.....	7
2.2.1	Les éléments	7
2.2.2	Les conteneurs d'éléments	7
2.3	Conception logiciel : extension pour le rendu	8
2.4	Conception logiciel : extension pour le moteur de jeu.....	8
3	Rendu : Stratégie et Conception	10
3.1	Stratégie de rendu d'un état.....	10
3.2	Conception logiciel.....	10
3.3	Conception logiciel : extension pour les animations	10
3.4	Ressources.....	10
3.5	Exemple de rendu	10
4	Règles de changement d'états et moteur de jeu	12
4.1	Horloge globale	12
4.2	Changements extérieurs.....	12
4.3	Changements autonomes	12
4.4	Conception logiciel.....	12
4.5	Conception logiciel : extension pour l'IA	12
4.6	Conception logiciel : extension pour la parallélisation.....	12
5	Intelligence Artificielle	14
5.1	Stratégies	14
5.1.1	Intelligence minimale.....	14
5.1.2	Intelligence basée sur des heuristiques.....	14
5.1.3	Intelligence basée sur les arbres de recherche	14
5.2	Conception logiciel.....	14
5.3	Conception logiciel : extension pour l'IA composée	14
5.4	Conception logiciel : extension pour IA avancée	14

5.5	Conception logiciel : extension pour la parallélisation.....	14
6	Modularisation.....	15
6.1	Organisation des modules.....	15
6.1.1	Répartition sur différents threads	15
6.1.2	Répartition sur différentes machines	15
6.2	Conception logiciel.....	15
6.3	Conception logiciel : extension réseau.....	15
6.4	Conception logiciel : client Android	15

1 Présentation générale

1.1 Archétype

L'objectif de ce projet est de réaliser un jeu que l'on peut trouver sur internet sous le nom de World Wars, basé sur l'archétype du Risk, mais en s'éloignant de l'approche militaire du jeu et en rendant le thème plus léger. Ainsi, au lieu d'une bataille entre missiles et tanks, notre jeu mettra en scène le combat entre une armée de licornes et une armée de dragons.

1.2 Règles du jeu

Chaque joueur est le dirigeant d'une armée : l'un possède une armée de licornes, l'autre une armée de dragons. Le but du jeu est de conquérir tous les territoires ennemis.

Au début du jeu, les armées de deux joueurs sont égales en quantité, mais les territoires possédés par chaque armée et le nombre de soldats présents sur chaque territoire sont placés aléatoirement.

Le joueur actif choisit le territoire qui attaque ainsi que le territoire ennemi qu'il veut attaquer, tout en sachant qu'il ne peut attaquer qu'un territoire frontalier. Un lancer de dés a alors lieu : chacun des deux joueurs a autant de lancers de dés qu'il a de soldats sur son territoire. Les résultats des lancers de chaque joueur sont ensuite sommés.

Si celui qui fait le score le plus haut est l'attaquant, alors celui-ci devient propriétaire du territoire ennemi et tous les soldats du territoire attaquant sauf un migrent vers le territoire nouvellement conquis, offrant un territoire en plus pour l'attaquant et une nouvelle répartition des soldats.

En revanche si celui qui fait le score le plus haut est le défenseur alors le territoire attaquant perd tous ses soldats sauf un.

Lorsque le joueur considère qu'il a fini son tour, il passe la main à l'autre joueur.

1.3 Ressources

Pour réaliser ce projet, nous utiliserons les trois textures suivantes :



Figure 1 – Textures des « soldats »

1	2	3
4	5	6
7	8	

Figure 2 – Textures des bulles indiquant le nombre de soldats sur un territoire



*Figure 3 –
Textures des différents territoires : inaccessible, occupé par les dragons, occupé par les licornes*

2 Description et conception des états

2.1 Description des états

Un état du jeu est formé d'éléments que l'on a divisé en deux catégories : les éléments fixes et les éléments variables. Les territoires (ou cellules) du plateau sont des éléments fixes tandis que l'armée occupant un territoire et le nombre de soldats présents sur ce territoire sont des éléments variables. Tous les éléments possèdent des coordonnées (x,y) permettant de les repérer sur le plateau et un identifiant qui indique sa nature (fixe ou variable).

2.1.1 Eléments fixes

Le plateau du jeu est formé par un ensemble de cellules de forme hexagonale représentant les territoires pouvant appartenir à l'un ou l'autre des joueurs.

Certains territoires sont inaccessibles : aucun des deux joueurs ne peut envahir ce type de territoire. Ils ont pour but de permettre la modification du plateau à chaque partie différente et de complexifier légèrement le jeu en constituant des obstacles. Ils sont représentés par une cellule avec des montagnes.

Le reste des territoires appartient forcément à l'un ou à l'autre des joueurs. Si le territoire appartient aux licornes, il est bleu ; s'il appartient aux dragons, il est rouge.

2.1.2 Eléments variables

Selon l'évolution du jeu, un territoire – à partir du moment où il est accessible – peut appartenir soit à l'armée des licornes, soit à l'armée des dragons. Les données concernant l'armée qui occupe un territoire et le nombre de soldats présents sont variables.

En effet le nombre de soldats évolue forcément lors d'une attaque et peut aller de 1 à 8 soldats, et après un combat un territoire des licornes peut devenir propriété des dragons et inversement.

2.1.3 Etat général

L'état général est décrit par deux tableaux, l'un contenant les éléments fixes (les territoires) et l'autre les éléments variables (le type de l'armée et son nombre). Ces deux tableaux permettent de décrire l'état du jeu à un instant donné.

2.2 Conception logiciel

Le diagramme des états est présenté en Figure 4. On distingue deux types de classes : les éléments et les conteneurs d'éléments.

La sémantique des couleurs utilisées est la suivante :

- Le rose symbolise les classes d'éléments. Le rose le plus foncé correspond à la classe mère, abstraite ; le rose plus clair correspond aux deux classes filles
- Le vert a quant à lui été utilisé pour les classes de conteneurs d'éléments

2.2.1 Les éléments

Nous avons trois classes : la classe Element, la classe Territory et la classe Team.

La classe Element est la classe mère des deux autres classes. Elle contient deux attributs qui sont les coordonnées x et y de chaque élément dans la grille du plateau de jeu, un constructeur et un destructeur, et une méthode nommée isStatic(). Cette dernière méthode est abstraite, et nous permet de savoir si un élément est variable ou fixe ; nous avons créé cette méthode car il n'y a pas de possibilité d'introspection d'une instance en C++.

La classe Territory a été créée pour représenter les cellules constituant les territoires du jeu. Elle possède un unique attribut, territoryStatus, dont la vocation est d'indiquer si la cellule peut être occupée par une armée ou non : l'attribut peut valoir accessible ou impossible ; ainsi qu'un constructeur, un getter pour obtenir la valeur de l'attribut territoryStatus et la redéfinition de la méthode isStatic().

La classe Team sert à représenter l'armée. Elle a donc deux attributs. L'un indique de quelle armée il s'agit : c'est l'attribut teamStatus, dont les deux valeurs possibles dragons et unicorns ; l'autre, nommée nbCreatures, donne le nombre de soldats occupant la case : un entier entre 1 et 8. Elle possède également quelques méthodes : un constructeur, les getters et setters de teamStatus et nbCreatures et la redéfinition de la méthode isStatic().

2.2.2 Les conteneurs d'éléments

On distingue la classe ElementTab de la classe State.

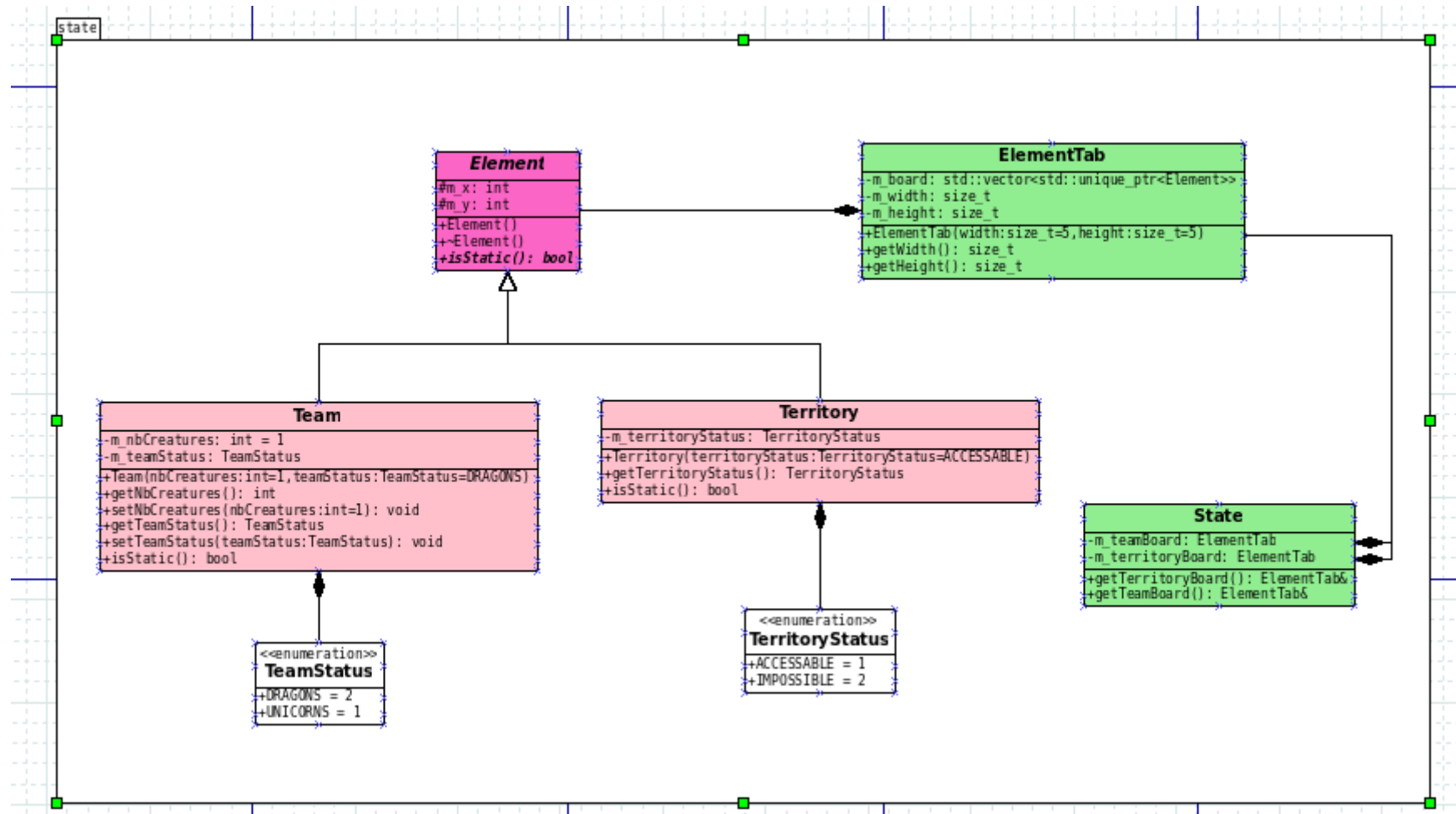
La première nous permet de construire des tableau à deux dimensions contenant des objets de type Element, offrant la possibilité de créer notre plateau de jeu à base des cellules de type Territory.

La classe State permet quant à elle d'accéder à la totalité des données de l'état.

2.3 Conception logiciel : extension pour le rendu

2.4 Conception logiciel : extension pour le moteur de jeu

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'implémentent 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 liés 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 une 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

