Projet Logiciel Transversal

Adou DIALLO – Raphaël DUJARDIN

Table des matières

1 Objectif

1.1 Présentation générale

Le but du projet est la réalisation d'un jeu de stratégie tour par tour permettant la conquête de territoires inspiré du jeu éponyme RISK avec un nombre moindre de règles beaucoup plus simples.

1.2 Règles du jeu

1.2.1 Aperçu

Le but du jeu est de pouvoir s'emparer des territoires adverses tout en défendant le sien et tous ceux précédemment conquis.

- Univers

L'univers du jeu est une carte 2D en vue de dessus. La carte est décomposée en territoires et en mers. On dénombre 5 types de territoires : prairie, montagne, forêt, désert, et côtier, repérables par une texture et une légende sur laquelle les joueurs se déplaceront. A cela s'ajoute un type particulier de territoire qui est le territoire neutre. Chaque joueur possède un territoire capitale qui a une grande importance. Sur tous ces territoires excepté le neutre sont présentes des troupes.

- Les joueurs

Les joueurs peuvent être au maximum au nombre de 6. Ils ont chacun une couleur définie qui servira à différencier les territoires possédés par chacun.

- Les personnages

Les personnages présents sur la carte qui permettront au joueur de jouer seront les troupes qui serviront à défendre ou conquérir un nouveau territoire. Parmi ces troupes existe un héros qui est un soldat particulier avec des caractéristiques différentes des troupes communes.

- Les items

Les items apparaissent de manière aléatoire au bout d'un certain nombre de tours. Il y en a de plusieurs sortes avec des effets plus ou moins puissants. Ces objets peuvent autant être des malus que des bonus pour le joueur qui devra dès lors bien réfléchir à leur utilisation.

- Les actions

Il y a deux types d'actions principales dans le jeu : attaquer et défendre. On peut aussi construire des ports. Il y a possibilité de passer son tour mais des mesures adéquates apparaissent à la prise de cette décision.

- Les tours

Chaque joueur joue chacun son tour, pendant un tour il peut y avoir plusieurs phases qui peuvent être influencées par la présence ou non d'item.

1.2.2 Règles détaillées

- Au début de la partie, tous les territoires sont neutres et chaque joueur choisit à tour de rôle un territoire qui sera sa capitale. S'il la perd, il meurt et celui qui a conquis sa capitale conquiert automatiquement tous ses autres territoires.
- Chaque joueur joue chacun son tour. Chaque joueur peut à tout moment de son tour décider de passer la main au joueur suivant sans terminer son tour. Chaque joueur peut à tout moment de son tour décider d'abandonner, dans ce cas il est éliminé de la partie et tous ses territoires redeviennent neutres avec les soldats présents dessus, et peuvent donc être capturés par un autre joueur.
- Pendant un tour, plusieurs phases successives :
  - Renforts : le joueur choisit un de ses territoires sur lequel il reçoit ses renforts (3 soldats par territoire contrôlé)
  - Action (capturer / attaquer / construire un port) : le joueur choisit un de ses territoires

comme territoire de départ, et un territoire comme territoire d'arrivée. Si le territoire d'arrivée est le même que le territoire de départ, un port y est construit. Si le territoire d'arrivée est un territoire neutre (contrôlé par aucun autre joueur), il est capturé ainsi que les soldats présents dessus. Si le territoire d'arrivée appartient à un autre joueur, une bataille est livrée. Un port ne peut être construit que si le territoire n'en est pas déjà doté. Un territoire adverse ne peut être attaqué que si la puissance d'attaque totale est supérieure à la puissance de défense totale. La capture de territoire neutre et l'attaque de territoire adverse ne peut se faire que sur un territoire adjacent ou accessible par la mer si le territoire de départ est doté d'un port.

↘ Déplacement : le joueur choisit un de ses territoires comme territoire de départ et un autre de ses territoires comme territoire d'arrivée, la moitié des soldats présents sur le territoire de départ est alors déplacée sur le territoire d'arrivée. Là encore soit les deux territoires doivent être adjacents soit ils sont séparés par une mer et le premier doit être doté d'un port.

On peut attaquer plusieurs pays dans le même tour mais les dégâts sont d'autant moindres.
Chaque territoire a un type (prairie/montagne/forêt/désert/côtier), répartis par zones de plusieurs territoires, quand on a 4 types de territoires différents on débloque le héros qui a beaucoup de puissance mais si il meurt toutes les troupes sont /2 sur tous les territoires
le héros représente 20 % de toutes les troupes qu'on possède en tout en terme de puissance

Par ci par là il y a quelques territoires qui restent neutres toute la partie et ne peuvent jamais être capturés. Ils ne peuvent jamais être attaqués. On peut déplacer des troupes dessus mais pas plus de 3 tours sinon elles disparaissent (le tour où on vient, un tour supplémentaire où on peut rester dessus, et le tour où on doit repartir, au 4è tour ça a disparu). On ne peut ni défendre ni attaquer avec ces troupes tant qu'elles restent là (mais elles ne peuvent pas être attaquées et un autre joueur ne peut pas placer de troupes sur le territoire).
De temps en temps un objet apparaît sur un territoire neutre et on peut le ramasser. 10 % de chances que ce soit un malus sinon c'est un bonus. 5 % de chances que ce soit le fat bonus (temporaire). Certains objets peuvent être conservés et utilisés quand on veut. Au bout de 4 tours l'objet disparaît s'il n'est pas utilisé. On ne peut pas en ramasser un nouveau si on en a déjà un réserve.

Liste des items :
- attaque n'importe où (force normale)
- attaque autant de pays qu'on veut dans le tour mais tous à puissance max
- attaquer et construire un port dans le même tour
- jouer 2 tours d'affilée
- bonus de troupes (10% sur chaque territoire) (exécuté tout de suite)
- malus de troupes (10 % sur chaque territoire) (exécuté tout de suite)
- perte d'un territoire (au hasard mais pas la capitale ni le pays où le joueur a le plus de troupes) (exécuté tout de suite)
- fat bonus : on acquiert le 2è et le 3è plus gros pays en nombre de troupes de tous les autres joueurs (exécuté tout de suite)

L'objet apparaît aléatoirement.
Lieu : aléatoire complet (équiprobable) parmi les territoires neutres.
proba globale : 8,5 à 12,5 % (aléatoire flottant) (probabilité qu'un objet apparaisse dans le tour)
proba de type majeur :
85 % normal
10 % malus
5 % fat bonus

proba de type mineur :
- les 5 bonus normaux : 20 % chacun
- les 2 malus : 50 % chacun
- fat bonus : 100 %

à part pour le fat bonus, on ne voit pas à l'avance ce qu'est l'objet.

Quand on conquiert une capitale, on conquiert tous les autres territoires du joueur
quand on conquiert un territoire à port, on conserve le port

attaque/défense par la mer : puissance divisée par deux

génération de map aléatoire, pourcentage réglable de flotte/terre, taille de la map en fonction du nombre de joueurs

quand on attaque un pays, la puissance d'attaque c'est le nombre de troupes sur le pays d'où on attaque (/2 si par la mer), la puissance de défense c'est le nombre de troupes sur le pays attaqué + sur les pays adjacents (/2 si par la mer), on ne peut attaquer que si la puissance d'attaque est supérieure à la puissance de défense ; les dégâts sont (puissance d'attaque – puissance de défense) * facteur de chance. Le facteur de chance est tiré aléatoirement entre 0,5 et 1,5.

1.3 Conception Logiciel

Packages :
- state
- render
- engine

Dépendances :
- SFML 2

2 Description et conception des états

2.1 Description des états

L'état du jeu est caractérisé par :
- le joueur dont c'est actuellement le tour
- la phase actuelle au sein du tour (placement des renforts / action / déplacement)
- l'item qui a été activé pendant le tour, soit automatiquement soit sur action du joueur, le cas échéant
- l'état de chaque joueur et de chaque territoire

L'état d'un joueur est caractérisé par :
- s'il est en vie ou pas
- le territoire qui est sa capitale
- le territoire sur lequel est positionné son héros
- l'item qu'il a en inventaire et le nombre de tours pendant lesquels il l'aura encore avant disparition, le cas échéant

L'état d'un territoire est caractérisé par :

- la liste des territoires adjacents
- ses propriétés géométriques (position et forme), utile pour le rendu
- son type (prairie / forêt / montagne / désert / côte)
- s'il est prenable ou non
- le joueur qui le possède le cas échéant
- l'item présent dessus le cas échéant
- s'il possède des ports ou non
- le nombre de troupes présentes dessus

2.2 Conception logiciel

L'état du jeu sera représenté par trois classes :

- Game pour l'état du jeu global
- Player pour les joueurs
- Land pour les territoires.

Le cycle de vie des objets Player et Land dépend de la classe Game. La classe Game est par ailleurs un singleton car elle représente l'état global de la partie.

Les items ne seront pas représentés par une classe car chaque item a un comportement très différent qui peut intervenir à beaucoup de niveaux différents (dans beaucoup de méthodes différentes), le plus simple, y compris pour ajouter de nouveaux items, est de tester l'item et de faire son action directement dans les méthodes concernées.

Voir diagramme des classes

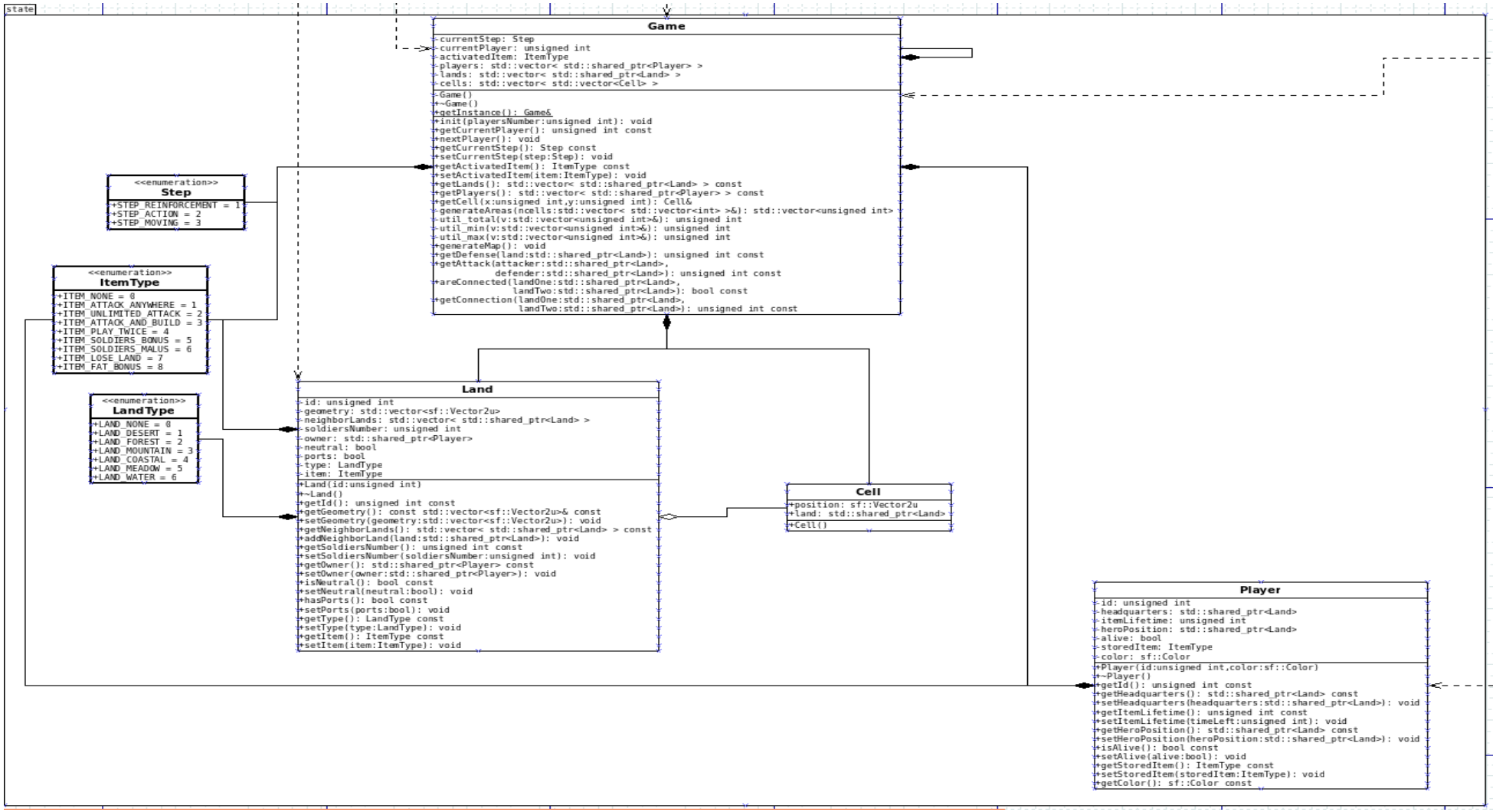
2.3 Conception logiciel : extension pour le rendu

La classe Land possède un attribut *geometry* de type liste de structures *Cell*. Chaque objet *Cell* possède une position x, y dans la grille des cellules de la carte (64x64 cellules), et des informations de bordure sous la forme d'un booléen pour chaque direction (haut, bas, gauche, droite). Les cellules auront une taille à l'écran de 9x9 pixels et les bordures feront 3 pixels d'épaisseur.

2.4 Conception logiciel : extension pour le moteur de jeu

2.5 Ressources

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

Tout l'état du jeu est accessible via le singleton Game, qui expose toutes ses données via des accesseurs. On peut donc construire une classe externe pour s'occuper du rendu.

Le rendu est effectué à 30 FPS, il n'a pas besoin d'être synchronisé avec le moteur de jeu puisque ce taux de rafraîchissement est largement supérieur à la vitesse de changement de l'état du jeu, sans consommer de ressources excessives.

Effectuer le rendu avec une classe externe à Game permet de séparer l'état de jeu de son affichage et de la gestion des ressources liées à cet affichage. Cette séparation sera bienvenue quand le jeu sera doté d'une architecture client / serveur.

3.2 Conception logiciel

Le package render est composé d'une unique classe Renderer, un singleton. Celui-ci fait appel au singleton Game du package state pour accéder aux données de l'état du jeu, et s'occupe de l'affichage ainsi que de la gestion des ressources (textures, polices, …).

La boucle événementielle est externe à cette classe Renderer. La classe Renderer a pour seul rôle de dessiner l'état du jeu. La gestion des événements ne lui reviendra pas, elle se fait donc pour le moment dans la fonction *main*. Renderer expose une méthode **void *render* (sf::RenderWindow&)** appelée dans la boucle événementielle pour le dessin du jeu.

3.3 Conception logiciel : extension pour les animations

3.4 Ressources

Numéro du joueur	**Couleur distinctive**	**Son distinctif**
1	Rouge (#ff0000)	res/sounds/player1.wav
2	Cyan (#00ffff)	res/sounds/player2.wav
3	Jaune (#ffff00)	res/sounds/player3.wav
4	Violet (#8000ff)	res/sounds/player4.wav
5	Noir (#222222)	res/sounds/player5.wav
6	Rose (#ff00ff)	res/sounds/player6.wav

Type de territoire	Texture
Prairie	Pas de texture
Forêt	res/textures/forest.jpg
Montagne	res/textures/mountain.jpg
Désert	res/textures/desert.jpg
Côtier	res/textures/coastal.jpg

Chaque territoire sera rendu avec une texture selon son type, filtrée par une couleur selon le joueur qui possède le territoire. Les territoires neutres auront la couleur blanche.

Les étendues d'eau n'auront pas de texture mais seront de couleur bleu (#0000ff).

Les ports seront matérialisés par une série de carrés noirs (représentant des pontons) disposés le long du littoral du territoire concerné, côté eau.

3.5 Exemple de rendu

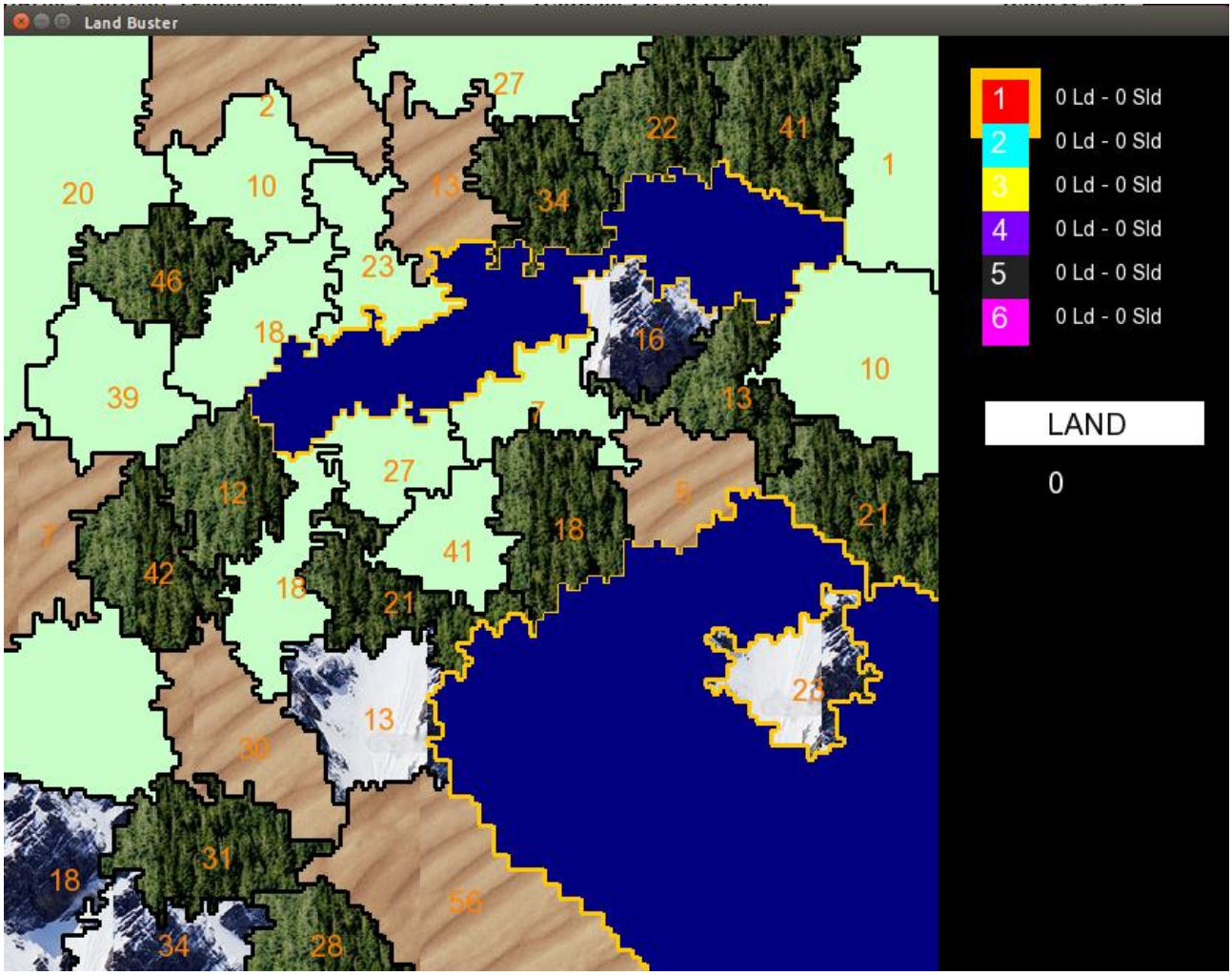
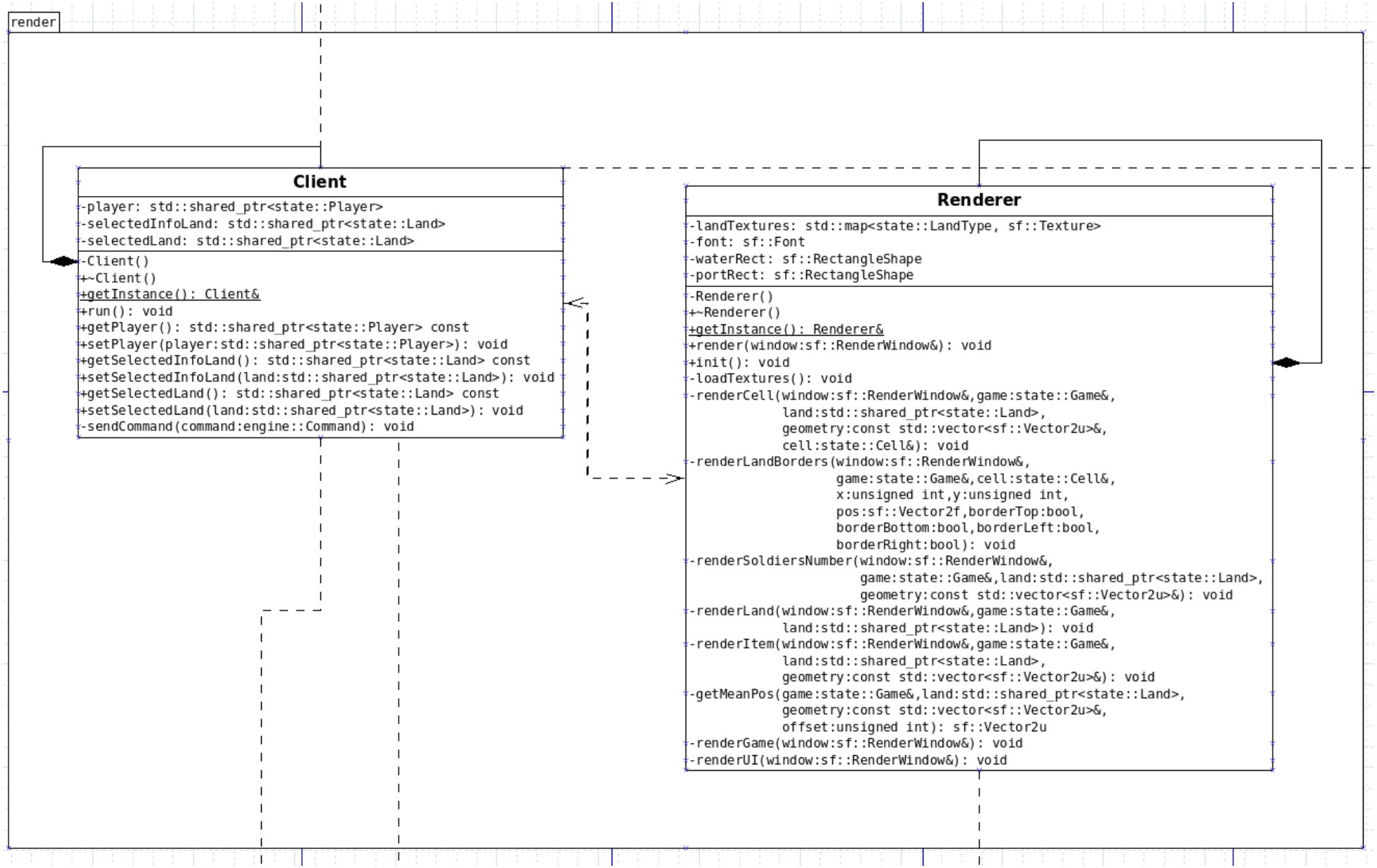


Illustration 2: Diagramme de classes pour le rendu



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

Le jeu est purement tour par tour, son état ne dépend absolument pas du temps, il n'y a donc aucune horloge.

4.2 Changements extérieurs

Chaque joueur peut en permanence accéder aux statistiques joueurs (liste des joueurs avec leur numéro, leur couleur, le nombre de territoires qu'ils contrôlent et le nombre de soldats que cela totalise, ainsi que le joueur en train de jouer). Chaque joueur peut également en permanence demander des informations sur un territoire donné : joueur qui possède le territoire le cas échéant, nombre de soldats présents et donc pouvant être capturés dans le cas d'un territoire neutre, sinon défense totale et attaque totale si le territoire appartient à un autre joueur. La défense totale représente la capacité de défense du territoire (nombre de soldats présents sur le territoire et sur les territoires adjacents ou accessibles par la mer appartenant au même joueur), l'attaque totale représente la capacité d'attaque du joueur sur le territoire (nombre de soldats présents sur le territoire d'où il attaque). La requête de ces informations n'est pas un changement d'état, c'est une action qui ne concerne que le joueur lui-même et qui sera gérée uniquement du côté IHM. Toutefois la classe d'état Game exposera les méthodes adéquates pour récupérer ces informations, ce qui permettra par exemple à une IA d'y requérir également.

Les changements extérieurs qui peuvent survenir sont :
- choix d'un territoire capitale (en début de jeu) / paramètres : numéro du joueur, territoire concerné
- choix d'un territoire où apporter les renforts / paramètres : numéro du joueur, territoire concerné
- construction d'un port sur un territoire / paramètres : numéro du joueur, territoire concerné
- passer son tour / paramètres : numéro du joueur
- abandon d'un joueur / paramètres : numéro du joueur
- attaque d'un territoire (si le territoire est neutre il est capturé, sinon une bataille est livrée) / paramètres : numéro du joueur qui attaque, territoire d'où il attaque, territoire qu'il attaque
- déplacement de soldats d'un territoire à l'autre / paramètres : numéro du joueur, territoire d'origine, territoire de destination
- utilisation d'un item en attente / paramètres : numéro du joueur

L'état de jeu contient les informations sur le joueur dont le tour est en cours et la phase de tour dans laquelle il est, ce qui limite les changements licites. Seuls ceux-ci seront exécutés par le moteur de jeu, et celui-ci incrémentera de manière adéquate la phase ou le joueur courant.

4.3 Changements autonomes

Les changements autonomes sont :
- expiration d'un item en attente
- apparition d'un item sur un territoire

En outre, des changements peuvent être conséquence d'un changement extérieur :

- victoire/défaite d'un joueur

4.4 Conception logiciel

Le jeu est divisé en deux grandes parties :

- le client, qui gère la conversion des entrées utilisateur (événements SFML) en commandes pour le moteur de jeu, et qui gère le rendu de l'état de jeu
- le moteur de jeu, qui gère la vérification de la validité des commandes et qui les exécute en modifiant l'état de jeu

Les deux parties ont en commun l'état de jeu, mais seul le moteur agit dessus, le client ne fait que le lire pour effectuer le rendu.

Le gameplay est tel que toute action qu'on puisse faire concerne toujours simplement un à deux territoires. On peut donc décrire une commande par une simple classe composée de deux identifiants de territoires. Comme il est prévu par la suite de placer le moteur de rendu et le moteur de jeu dans deux threads séparés, puis avec chacun sa version de l'état du jeu, nous avons décidé de désigner les éléments de jeu par un identifiant numérique plutôt que par un pointeur direct.

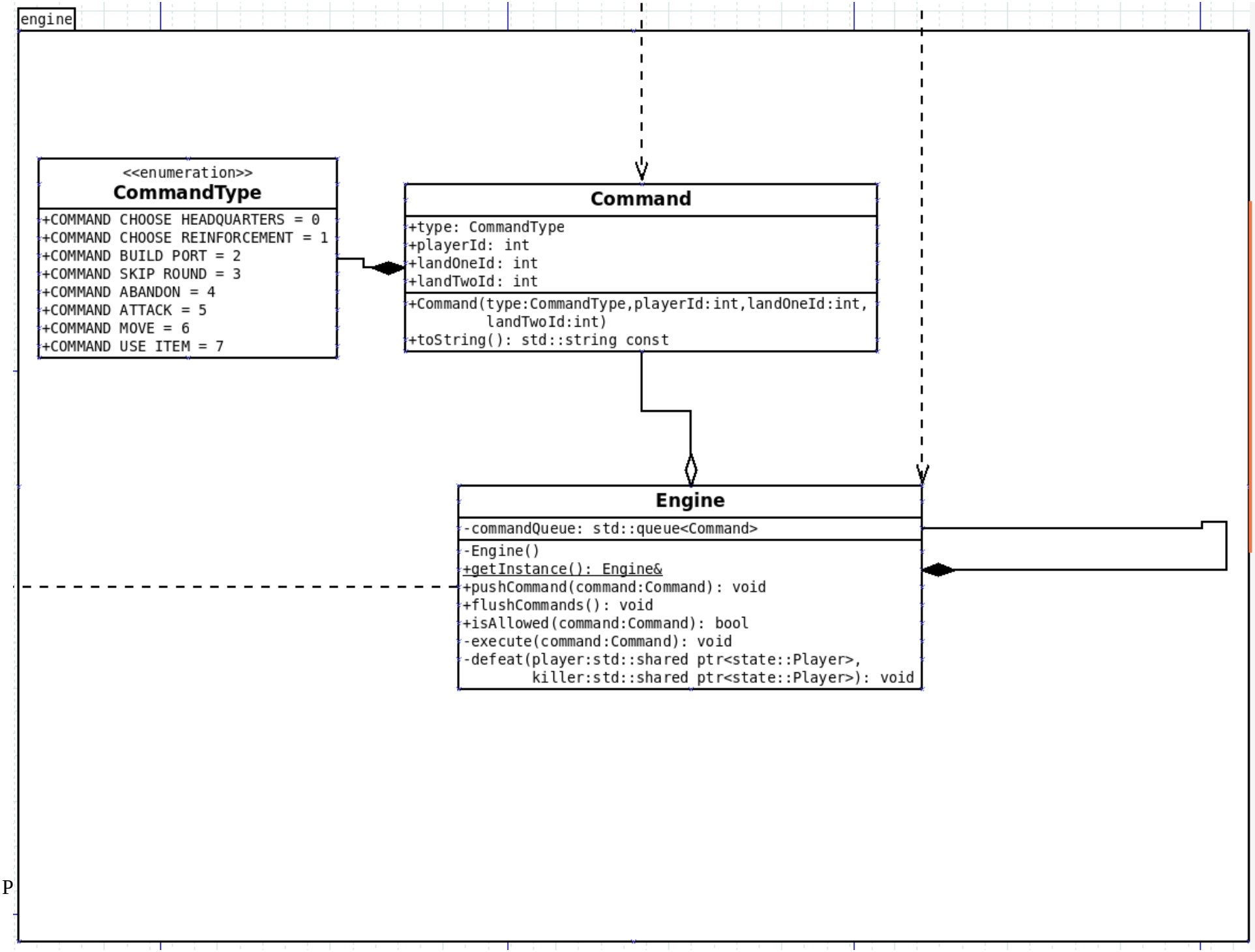
4.5 Conception logiciel : extension pour l'IA

La séparation des événements clavier/souris et des commandes pour le moteur de jeu permettra d'insérer des joueurs IA, qui enverront directement des objets de notre classe Command au moteur de jeu comme le ferait le client d'un utilisateur humain.

4.6 Conception logiciel : extension pour la parallélisation

Comme dit précédemment, nous avons désigné les éléments de l'état de jeu par un identifiant numérique, une solution indépendante de la disposition en mémoire des éléments. Cela permettra de manipuler l'état de jeu sur plusieurs machines différentes.

Illustration 3: Diagrammes des classes pour le moteur de jeu



P

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

L'ensemble de la stratégie d'intelligence artificielle est décomposé en plusieurs niveaux d'intelligence : de la plus simple à la plus complexe, sachant que les niveaux supérieurs d'intelligence exploitent ce qui a été fait dans les niveaux inférieurs.

5.1.1 Intelligence minimale

Le niveau le plus simple d'intelligence artificielle proposé est comme suit :

-Tant qu'il y a un territoire limitrophe neutre on essaie de le récupérer.
-Construction aléatoire de ports.
-Le jeu se termine lorsque qu'il ne reste plus aucun territoire libre sur la carte.

5.1.2 Intelligence basée sur des heuristiques

Le but ici est d'offrir un comportement plus intelligent que celui basé sur le simple aléatoire :

-Construction moins aléatoire de ports

5.1.3 Intelligence basée sur les arbres de recherche

L'algorithme de MinMax sera l'algorithme sur lequel on se basera pour créer une intelligence plus avancée. L'IA devra être capable de faire le meilleur choix entre par exemple attaque et construire des ports. On a aussi un système d'évaluation des scores et une fonction de rollback qui permet de revenir en arrières, d'annuler certains mouvements (déplacement, construction de ports).

5.2 Conception logiciel

Classe AI. Toutes les formes d'intelligence artificielle implantent la classe abstraite *AI*. Le rôle de ces classes est de fournir un ensemble de commandes à transmettre au moteur de jeu. Notons qu'il n'y a pas une instance par joueur, mais qu'une instance doit fournir les commandes pour tous les joueurs. La classe *DumbAI* implante l'intelligence minimale, telle que présentée ci-dessus. De même, la classe *DeepAI* implante la version améliorée.

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