

Projet Logiciel Transversal

Antoine Delavoy pierre – Nicolas Leteinturier

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 but de ce projet est de programmer un jeu basé sur l'archétype du Risk. Le jeu aura pour but de respecter le cahier des charges:

- être un jeu en Multijoueur afin de pouvoir avoir des application serveur et programmer une IA
- le jeu doit inclure une IA avec 3 Niveaux de difficultés
- le jeu doit être un jeu à état, pour Stocker les données du jeu à tout moment
- le jeu doit pouvoir être mis en réseau.

Le Risk est un jeu de stratégie en tour par tour qui permet de jouer à plusieurs joueurs les uns contre les autres. Ceci inclut donc une possibilité de jouer contre un ordinateur. Le but du jeu étant d'élaborer une stratégie militaire afin de gérer des troupes et prendre le contrôle du monde.

1.2 Règles du jeu

Tout en se basant sur les règles de l'archétype, cette version du jeu va prendre certaines libertés. Comme par exemple l'implémentation de plusieurs types d'unités : des unités plus défensives, et des unités plus offensives qui permettent d'avoir un coefficient multiplicateur sur les jets de dés en fonction de la situation.

Au début de la partie:

- Une carte est générée composée de N tuiles
- La carte est divisée entre les joueurs dans la partie
- Un même nombre d'unités militaires sont attribuées à chaque joueur à répartir sur ses territoires.

En Partie chaque joueur joue tour par tour. Les tours se déroulent de la façon suivante:

- Renforts:
en fonction du nombre de territoires que possède le joueur, il reçoit des renforts à distribuer sur les territoires de son choix

- Technologie:

Si le joueur a assez de points d'expérience il peut investir ses points dans une technologie pour gagner un bonus

- Attaque:

si un joueur a plus d'une unité sur un territoire il peut attaquer un territoire voisin. Le combat se fera en fonction de jet de dés en appliquant les bonus technologiques

- Déplacement:

Pour finir le tour le joueur peut déplacer des unités d'un territoire A vers un territoire B si les territoires sont reliés par un territoire du joueur.

Fin de partie:

- Un joueur perd si il n'a plus de territoires.
- Un joueur gagne si il possède 90% des territoires

1.3 Conception Logiciel

Afin de pouvoir programmer le jeu. Le sprite sheet a été créé. Une carte du monde a été divisé en 14 territoires (le nombre a été limité pour simplifier le jeu) chaque territoire a été isolé afin de pouvoir le traiter individuellement au niveau de la couleur. En effet la couleur du territoire sur l'interface graphique sera déterminé par quel joueur possède le territoire. Afin d'éviter de devoir préparer les sprites sheet de chaque couleur. La carte globale sera une superposition de territoires coloré individuellement grâce a SFLM.

2 Description et conception des états

2.1 Description des états

L'état du jeu à tout instant est défini par la possession de chaque territoire par un unique joueur, et par le nombre d'unité que ce joueur possède sur chaque territoire. Ainsi que l'ordre de passage des joueurs, et le joueur actuel qui doit jouer. Ainsi à tout instant il est possible d'interrompre et reprendre le jeu uniquement grâce à :

- la liste de joueurs.
- la liste des pays avec le nombre d'unité dessus
- le joueur actuel

2.2 Conception logiciel

Pour définir ces états nous avons défini plusieurs classes.

Dans un Premier temps la classe GameState, qui contient le vecteur des joueurs donc l'ordre de passage, le vecteur des pays (ces vecteurs contiendront des pointeurs vers des objets définis dans les classes Player et Country), et le joueur actuel qui doit jouer.

La classe GameState gère les ajoutes et suppressions de joueurs soit lors de la création de la partie ou ultérieurement lors d'une défaite gérée par le moteur de jeu. Une autre méthode dans la classe permet de déplacer des unités.

La classe Player défini par un identifiant unique.

Les méthodes de cette classe permettent de gérer les renforts reçus par un joueur en fonction du nombre de territoire qu'il possède.

La classe Country contient les informations sur chaque territoire. Comme le nombre d'unité et leurs types sur un territoire (sous forme d'un vecteur de Type d'unité). Afin de pouvoir attaquer les pays voisins pour chaque territoire un vecteur contenant les pointeurs vers les pays voisins est défini. Lors de la phase d'attaque le moteur autorisera seulement les attaques sur les territoires voisins ennemis. Ainsi le Pays doit aussi contenir les informations sur le propriétaire du pays.

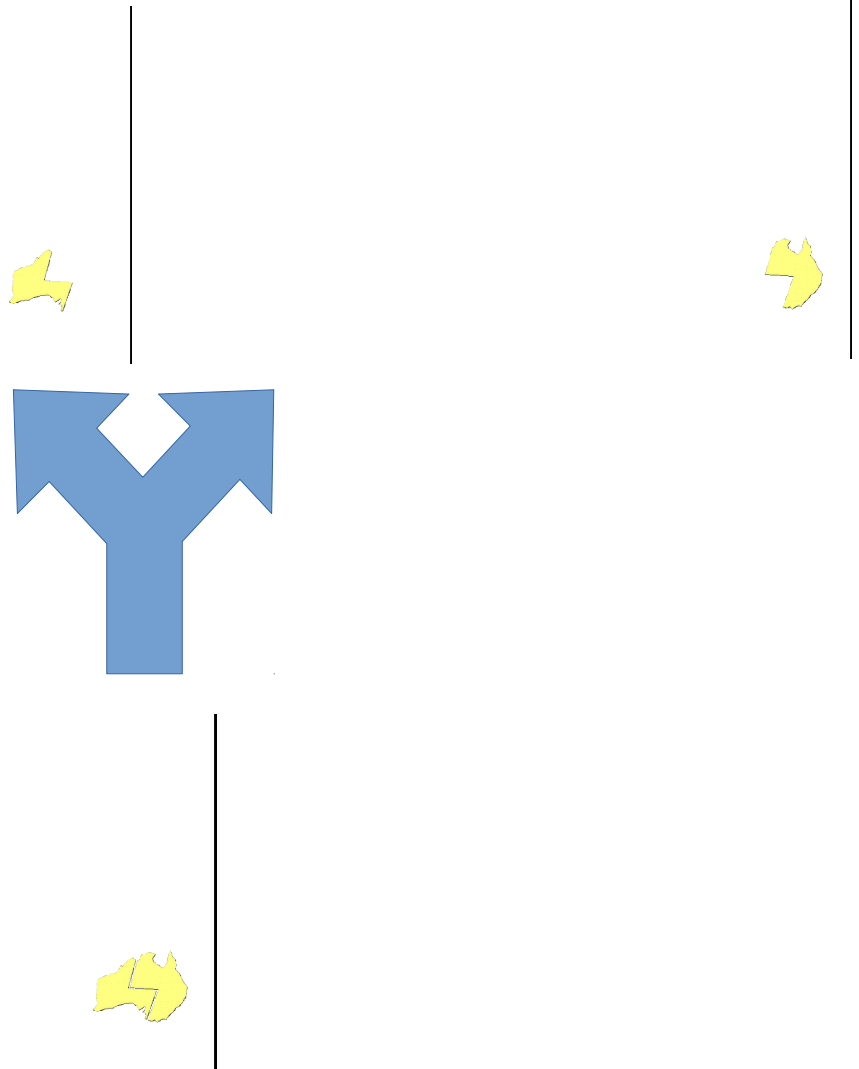
Les méthodes de la classe Country ont pour but de gérer les unités présentes sur le territoire. Soit ajouter des unités en cas de renfort. Éliminer des unités en cas de perte en attaque ou défense. Afin d'initialiser la carte et la liste des voisins, une méthode permet d'ajouter des territoires voisins.

La classe Unité et Type afin de gérer les différents types d'unités que le joueur pourra choisir pour élaborer une stratégie.

La Classe Action permet de gérer toutes les actions/envenimement du jeu, c'est à dire les Attaques, les mouvements, les renforts.... Lors d'une attaque par exemple les troupes des deux territoires impliqués sont enlevées temporairement des deux territoires pour être traitées dans la classe action. Si elle est victorieuse elles seront ainsi replacées à la fin dans le territoire où elles sont supposées aller. Si elles sont détruites une méthode les éliminera. Ainsi la classe Action permet de gérer quelle type d'action sera utilisé pour l'état de jeu. Les autres classes comme Attaque, mouvement, renforcement et initialisation vont hériter de la classe Action.

2.3 Conception logiciel : extension pour le rendu

Pour le rendu, la classe Country étant défini par un identifiant associé a chaque territoire. Le rendu général sera défini par une superposition des différents territoires. Pour un certain territoire il faut pour le rendu également établir un code couleur en fonction du propriétaire du territoire. Ainsi utiliser une superposition de masque indépendant permet de donner une couleur a chaque territoire plus facilement. Voir exemple de sprite dans le dossier ressources. Ici nous avons l'exemple de l'oceanie qui est divisé en deux terriories. Leur position respective sur la carte .png permet donc une superposition. Puis en appliquant un filtre de couleur grace au rendu nous pourrons avoir l'interface.



2.4 Conception logiciel : extension pour le moteur de jeu

Les classes ont été établies pour faciliter l'extension pour le moteur de jeu. La définition d'un territoire par un identifiant et une liste de pays voisins permet au moteur d'avoir un accès facile à

quel territoire un joueur peut attaquer. Avoir également un accès rapide pour chaque territoire au nombre d'unités dessus et au propriétaire permettra d'avoir plus d'informations aux quelles l'IA aura accès ultérieurement dans la programmation de l'IA.

2.5 Ressources

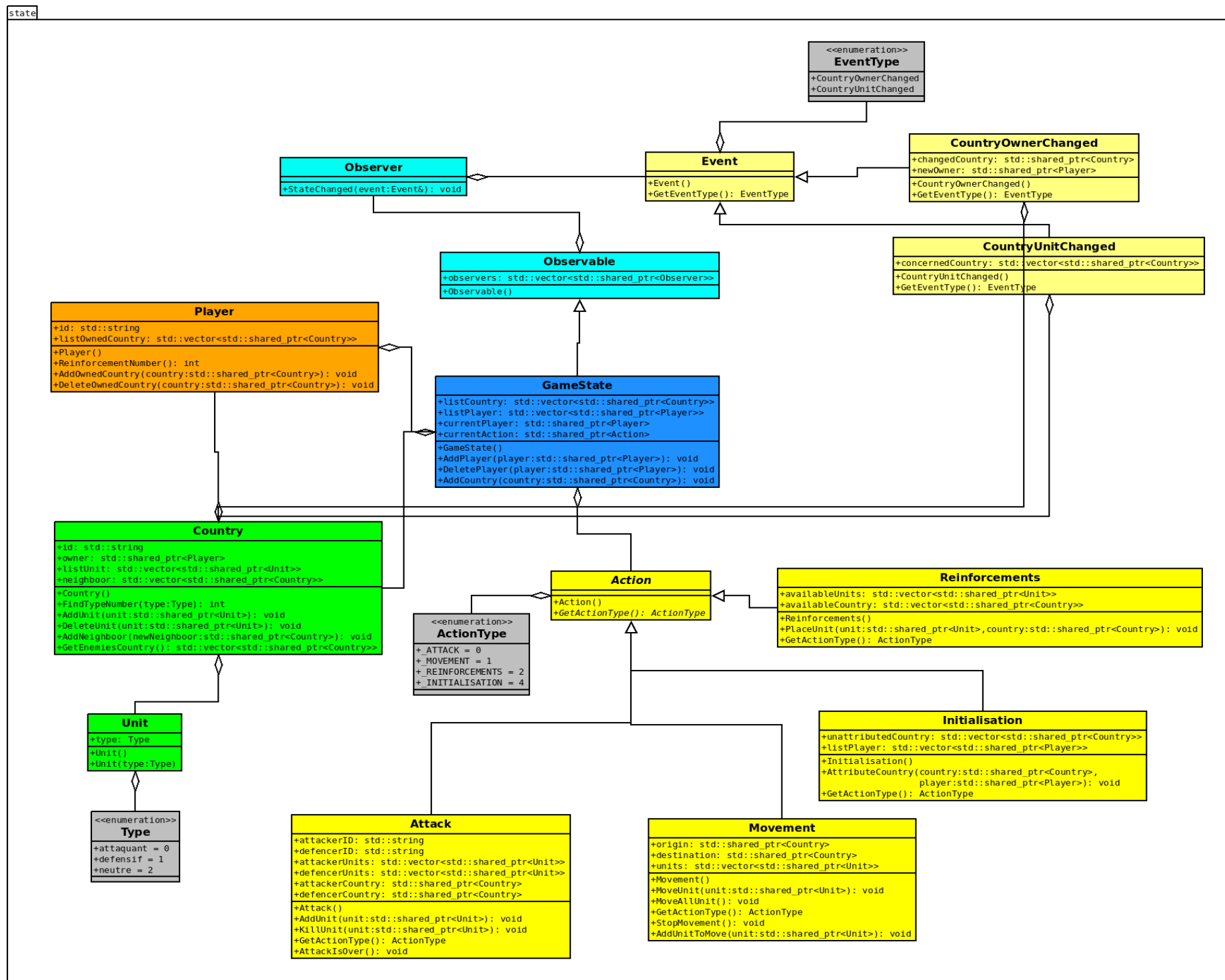


Illustration 1: Diagramme des classes d'état

3 Rendu : Stratégie et Conception

Afin d'obtenir un rendu nous avons décidé d'organiser notre diagramme de classe pour le rendu de la façon suivante :

Une classe Scène gèrera intégralité des données et de la synchronisation en ayant accès à l'état de jeu et aux autres classes. Scène pourra sélectionner quelle partie du rendu nécessite une mise à jour.

Les deux classes auxquelles ils peuvent faire appel sont WorldMap, et UnitRepresentation.

La classe WorldMap permet de gérer tout ce qui touche aux pays. C'est à dire la représentation des différents territoires. La classe importe donc dans un premier temps tous les sprites de pays ce qui permet de construire le monde. Une fois le monde créé cette classe permet également de mettre en couleur individuellement chaque pays afin de visualiser quel joueur est propriétaire du territoire.

La classe UnitRepresentation de son côté s'occupe d'extraire de l'état de jeu le nombre d'unités sur chaque territoire afin de le représenter sur la carte. Pour ce faire il faut placer au bon endroit les valeurs des nombres d'unités.

Toutes les positions et les adresses des sprites sont stockées dans un fichier « define.h » afin de ne pas avoir de code trop lourd avec des tableaux au milieu du code.

La classe Layer permet de stocker tous les sprites et textures des deux classes précédentes.

3.1 Stratégie de rendu d'un état

Les classes décrites ci-dessus permettent de traiter de façon indépendante le rendu des territoires et le rendu des unités. Ceci est important car le jeu est divisé en plusieurs phases. Dont des phases de renfort, où les territoires ne changent pas. Seul le nombre d'unités change. Ainsi en faisant ceci nous pouvons mettre à jour seulement la partie du script qui nous intéresse. Ceci nous permet de limiter la complexité en temps de calcul et plus de flexibilité.

3.2 Conception logiciel

Afin de donner accès aux informations nécessaires aux bonnes classes les deux classes WorldMap et UnitRepresentation héritent toutes les deux de Layer. Layer ayant accès à certaines données de l'état de jeu. Ceci permet de pouvoir représenter toutes les informations nécessaires à l'utilisateur pour une prise de décision. La dernière classe GraphicElement est une classe pour traiter chaque élément de Layer indépendamment. Ainsi on peut pour chaque pays modifier de façon indépendante ses autres sa couleur en fonction du propriétaire grâce à une simple méthode.

En parcourant la liste des pays issue de Scène la classe UnitRepresentation est capable de retrouver et représenter le nombre de chaque type d'unités sur chaque territoire.

3.3 Conception logiciel : extension pour les animations

3.4 Ressources

Nous avons pour cette partie ajouté quelques nouvelles ressources afin d'optimiser le rendu. Dans une première partie nous avons ajouté une représentation pour les différents types d'unités. Qui est représentée au centre de chaque pays. Nous avons donc dû déterminer la position de chaque pays afin de positionner ce rendu. Les positions sont sauvegardées dans le fichier define.h. Un fond de carte a été ajouté pour un aspect plus esthétique. Pour les différents territoires nous avons utilisé les

territoires décrit dans la partie précédente. Avec chaque territoire dans un fichier png. La carte est donc une superposition de tous les fichiers traité individuellement par le moteur de rendu.

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

Le moteur de jeux va régler les états de jeux : Reinforcements, Attack, Movement par une gestion par click sur le rendu graphique ou par un passage de tour en appuyant sur la barre espace. Ainsi le jeu va séquentiellement passer chaque phase de jeu pour chaque joueur.

4.2 Changements extérieurs

Afin de régler chaque phase de jeu le moteur de jeu va traiter chaque phase de jeu dans un module qui aura accès à l'état de jeu. Afin de gérer chaque phase de jeu indépendamment des interfaces spécifique à chaque phase de jeu ont été ajouté au rendu graphique pour choisir sur quelle pays doit être effectué l'action, puis le nombre d'unités impliqué.

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

