Projet Logiciel Transversal

WANG Zhao BENMIRA Eltadj PRADINES Louis

FIGURE 1 – Exemple du jeu

Table des matières

1 Présentation Générale

1.1 Archétype

L'objectif de ce projet est de réaliser un tactical RPG à la Fire Emblem, un exemple est proposé à la figure 1. Un tactical RPG est un jeu de rôle tactique. Dans ce genre de jeu vidéo, le gameplay est basé sur les décisions tactiques que le joueur doit prendre au cours des combats. Le joueur doit utiliser ses personnages pour éliminer tous les personnages ennemis du plateau, chaque personnage possède comme dans un RPG traditionnel un niveau et des statistiques qui définissent sa puissance.

1.2 Règles du jeu

1.2.1 Présentation des règles

Pour gagner la partie il faut tuer tous les personnages de l'ennemi (chaque joueur a 5 personnages). On joue au tour par tour, un tour est fini lorsque tous les personnages d'un joueur ont joué. Un personnage doit faire ces actions dans son tour :

1. se déplacer de [0 cases, maximum de distance parcourable]
2. attaquer si il y a un ennemi dans sa portée d'attaque xor utiliser une potion xor attendre

Chaque personnage possède des points de mouvement qu'il régénère au début de chaque tour, se déplacer sur une case consomme normalement 1 point de mouvement (ça dépend du type de case traversée). Lorsqu'il n'a plus de points de mouvements, un personnage ne peut plus se déplacer.

1.2.2 Plateau

Le plateau fait 17x17 cases, chaque cases possède des particularités

<table>
<tr><td rowspan="8">Plateau</td><td rowspan="8">Environnement</td><td>Plaine
(1 pm)</td><td colspan="3">C'est une zone plane et tous les personnages peuvent marcher dessus</td></tr>
<tr><td>Forêt
(2 pm)</td><td colspan="3">C'est une zone avec des arbres qui permet d'augmenter la chance d'esquiver les attaques pour tous les personnages</td></tr>
<tr><td>Pierre
(inf pm)</td><td colspan="3">C'est une zone interdite pour tous les personnages</td></tr>
<tr><td rowspan="2">Rivière
(inf pm)</td><td rowspan="2">C'est une zone interdite pour tous les personnages sauf l'assassin</td><td>Pont
(1 pm)</td><td>C'est un pont sur la rivière et tous les personnages peuvent le traverser sauf le chevalier</td></tr>
<tr><td>Passage
(3 pm)</td><td>C'est une partie moins profonde de la rivière, tous les personnages peuvent la traverser</td></tr>
<tr><td>Falaise</td><td colspan="3">C'est une case infranchissable d'un côté</td></tr>
<tr><td>Coffres</td><td colspan="3">Des coffres qui apparaissent aléatoirement sur le plateau, Il y a des armes et des potions dedans</td></tr>
</table>

1.2.3 Saisons

Il y a quatre saisons à la fin de chaque tour (lorsque les 2 joueurs ont joué) la saison change. Chaque saison donne des bonus généraux à tous les personnages, des bonus spécifiques aux personnages qui sont reliés à celle-ci et des malus aux personnages qui sont reliés à la saison opposée. L'ordre des saisons est le suivant : Printemps → Été → Automne → Hiver

Saison	**Saison opposée**	**Bonus ou malus généraux**
Printemps	Automne	Tous les personnages se soignent de 5 pv
Été	Hiver	Les personnages augmentent de 50% leurs pm
Automne	Printemps	Les forêts augmentent 2 fois plus l'esquive des personnages
Hiver	Été	Toutes les cases coûtent 2 fois plus de pm à traverser et toutes les rivières sont gelés et traversables

1.2.4 Personnages

Chaque joueur dispose de 5 personnages (Assassin, mage, chevalier, archer et combattant). Ces personnages possèdent des armes et des spécificités liées à la saison auxquelles ils sont reliés :

Assassin	Automne
Mage	Printemps
Chevalier	Hiver
Archer	Ete
Combattant	Neutre

Lorsque la saison d'un personnage est effective, il reçoit des bonus qui améliorent ses attaques et sa défense ainsi que ses déplacements. Chaque personnage possède ses propres armes et bonus lié à sa particularité :

L' assassin : L'assassin possède une dague, il traverse les cases rivières, et un bonus est appliqué à son attaque lorsqu'il attaque par derrière. En automne sa technique est multipliée par 2. Au printemps sa technique est divisée par deux.

Le Mage : Le mage utilise la magie, il peut apporter des soins. Au printemps le mage a la capacité de ressusciter le dernier personnage tué de son équipe. En hiver le mage perd ses pouvoirs de soin, il peut donc plus soigner aucun personnage.

Le chevalier : Le chevalier possède pour arme une épée, c'est un cavalier il a donc un grand nombre de points de déplacement. Il a pour particularité de pouvoir se déplacer après avoir attaqué (s' il lui reste des points de mouvements) mais il a pour malus de ne pas pouvoir traverser les ponts. En hiver le chevalier à la capacité d'augmenter sa défense. En été, la défense du chevalier est divisée par deux .

Archer : l'archer possède pour arme un arc et des flèches, il a pour spécificité de pouvoir augmenter sa portée d'attaque lorsqu'il est sur une case montagne. En été sa portée augmente mais en hiver elle baisse.

Combattant : Le combattant a pour arme une lance par défaut au début du jeu mais il a la capacité contrairement aux autres personnages de pouvoir porter toutes les armes possible. Il est considéré comme un personnage 'neutre ' il n'a donc pas de spécificité lié aux saisons. Le combattant possède un bonus qui lui permet d'obtenir deux fois plus d'armes à l'ouverture d'un coffre.

1.2.5 Statistiques

Chaque personnage possède des statistiques qui augmentent aléatoirement dès qu'il gagne un niveau

Vie	La vie du personnage, lorsqu'il a 0 pv il meurt
Niveau	Le niveau du personnage
Expérience	Lorsque un personnage a 100 d'expérience il gagne 1 niveau Elle augmente lorsque le personnage, attaque, se fait attaquer, tue un personnage, ou soigne un personnage
Points mouvements	Se déplacer sur une case coûte un certain nombre de pm
Force	Augmente les dégâts infligés par les armes non magiques
Magie	Augmente les dégâts infligés par les armes magiques
Technique	Augmente les chance d'infliger des critiques et les chances d'esquiver une attaque
Vitesse	Si un personnage à 3 de vitesse en plus que sa cible, elle attaque 2 fois
Défense	Diminue les dégâts physiques reçus
Résistance	Diminue les dégâts magiques reçus

1.2.6 Objets

Objets	Armes (portée)	Tomes (1-2)	Armes magiques (Un tome de soin peut soigner les alliés)
		Lance (1)	Précision élevée Fort contre l'épée
		Hache (1)	Gros dégâts de base Fort contre la lance
		Epée (1) Dague (1)	Armes rapides Fort contre la hache
		Arc (2-3)	Perd de la précision avec la distance
	Potions	restaure 10 pv	
	Échangeables	Les personnages peuvent échanger leurs objets avec les personnages à proximité	

1.3 Ressources

On a besoin de textures pour les cases, une case est un carré de 128 pixels. Les textures des cases de type herbe changent de couleur en fonction des saisons Chaque personnage a un modèle carré de 128 pixels qui le représente sur le plateau et un portrait plus détaillé qui le représente dans l'écran de statistiques, dans les dialogues,...

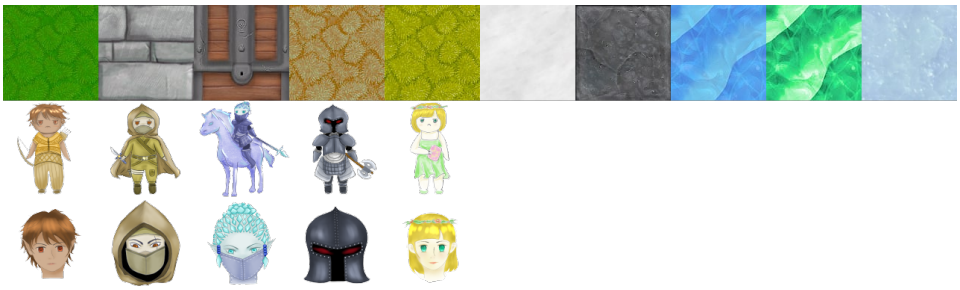


FIGURE 2 – Les textures du jeu

La police du jeu est la suivante

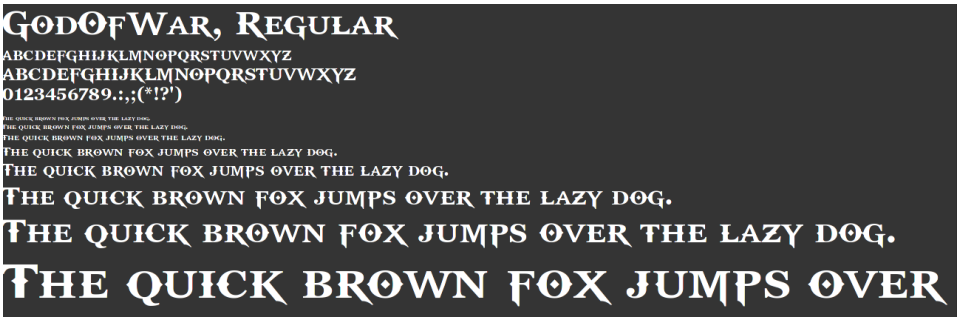


FIGURE 3 – La police du jeu

1.4 Simplifications

Nous sommes un groupe de 3 et rapidement nous avons accumulé du retard sur la conception du jeu. Ainsi pour rattraper notre retard nous avons simplifié les règles du jeu. Tout d'abord les cases n'ont plus de particularités, ie l'assassin ne peut plus traverser les cases rivières. De plus nous avons choisi de supprimer les cases Forêt, Pierre, Falaise et Coffre. Nous avons aussi choisi de ne pas implémenter les objets (armes et potions) et les capacités, ainsi chaque personnage a une portée d'attaque de 1. Pour finir nous avons aussi simplifié les saisons, les saisons accordent juste un bonus (resp. un malus) de 1 en force, technique, magie, vitesse, défense et résistance au personnage qui est de la même saison (resp. au personnage qui est de la saison opposée) et en hiver le coût en pm des cases est doublé mais on peut traverser la rivière gelé.

Les joueurs peuvent avec chaque personnage réaliser les combinaisons d'actions suivantes :

— déplacer puis attaquer
— déplacer puis attendre
— attaquer
— attendre

Si un personnage attaque un autre personnage ou attend c'est au tour du personnage suivant. Lorsque le joueur a joué avec tous ses personnages (qui sont automatiquement sélectionnés dans l'ordre par le state) c'est au tour de l'autre joueur dès que plus aucun personnage ne peut jouer, on passe au tour suivant.

Désormais le jeu se joue au clavier, les touches sont les suivantes :

— A : attendre
— Z,Q,S,D : se déplacer
— UP, DOWN, LEFT, RIGHT : attaquer

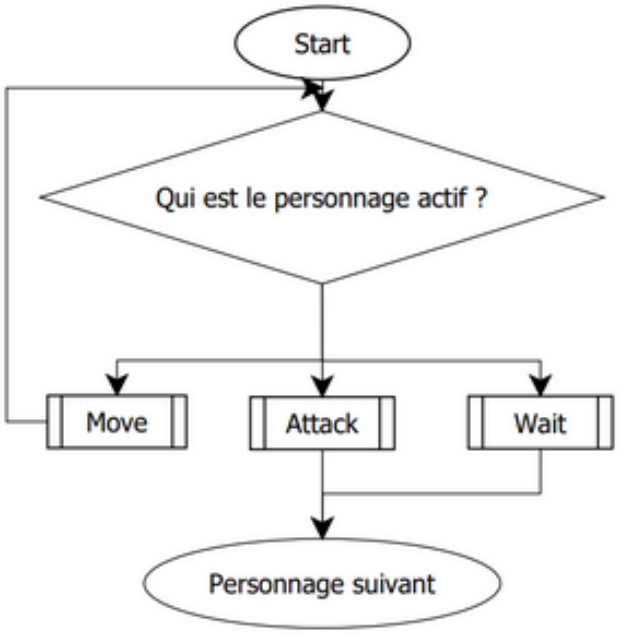


FIGURE 4 – Le déroulement d'une phase de jeu

2 Description et conception des états

Un état du jeu est formé par un ensemble d'éléments. Tous les éléments possèdent un identifiant, un Id qui permet de connaître leur type. Cet Id permettra notamment d'afficher chaque élément.

Remarque : L'id de chaque élément possède 2 chiffres, le chiffre de la dizaine indique la catégorie de l'élément, ainsi si c'est un joueur il commence par un 4, par 3 si c'est une saison, par 2 si c'est un objet, par 1 si c'est une case du plateau (cell) et par 0 si c'est un personnage. Le deuxième chiffre indique plus en détail dans quel état est l'élément.

2.1 Description des états

2.1.1 États des personnages

Chaque personnage possède une classe, il peut être :
— un archer
— un assassin
— un mage
— un chevalier
— un combattant

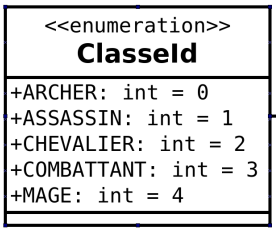


FIGURE 5 – les Id des classes

On associe à chacune de ces classes un Id qui lui est propre (attention plusieurs personnages peuvent avoir une même classe)

Un personnage possède aussi des statistiques qui lui sont propres et qui sont stockés dans une classe Statistiques. A chaque personnage on associe 2 instances de Statistiques : une pour les statistiques brutes (statistiques avant bonus ou malus) et une pour les statistiques actuelles (après application des bonus et des malus). On remarque aussi que la classe statistique contient des attributs statistiques maximum. Ils représentent la valeur que ne peut pas dépasser les statistiques "consommables" (ie la vie, les points de mouvements et l'expérience) lorsque le personnage accomplit une action qui les influence comme par exemple se soigner en utilisant une potion.

Uniquement 3 statistiques affectent l'état du personnage. Tout d'abord, la vie, en effet lorsque la vie du personnage atteint 0, celui-ci meurt et son état alive passe alors à false. Ensuite il y a les points de mouvements lorsqu' ils atteignent 0 le personnage ne peut plus se déplacer et enfin l'expérience, lorsque celle-ci atteint sa valeur maximale (définie par l'attribut experience_max) le personnage gagne un niveau. Les autres statistiques affectent les performances du personnage en combat.

En tout un personnage possède 8 statistiques influencés par les niveaux lorsqu' un personnage gagne un niveau, il a une probabilité x d'augmenter dans la statistique i. Ces probabilités sont définies par un vecteur de float : probaGainStats. Ce vecteur est propre à chaque classe.

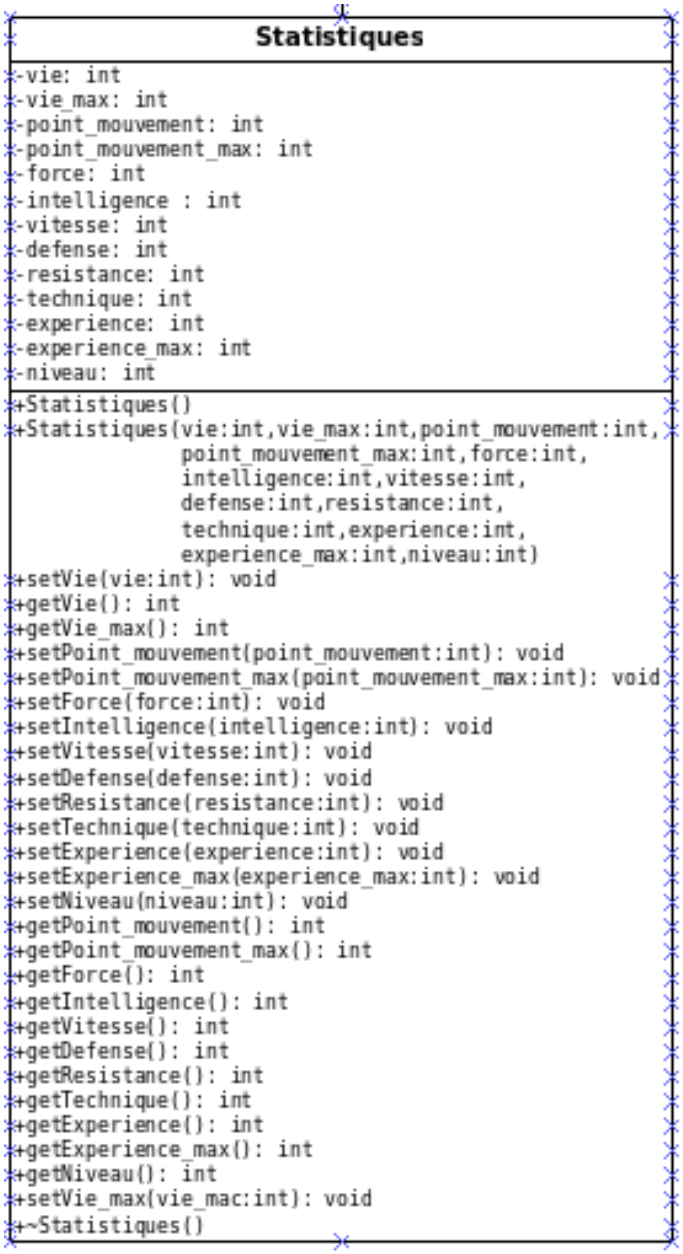


FIGURE 6 – la classe Statistiques

2.1.2 États des cases

Chaque case (cell) possède un type, elle peut être :
— de type grass (printemps, été, automne, hiver)
— de type river (hiver, autre)
— de type bridge
— de type passage

Chaque type a une texture qui lui est associée lors du rendu, de plus les types modifient aussi le coût en point de mouvement que coûte la case à traverser. Ce coût varie en fonction des saisons. De plus on peut remarquer que les types grass et river changent en fonction des saisons. Pour les cases de type grass c'est un indicateur visuel pour que le joueur sache quelle est la saison du tour actuel. Pour les cases de type river, c'est pour montrer que en hiver les rivières sont gelés et donc traversables.

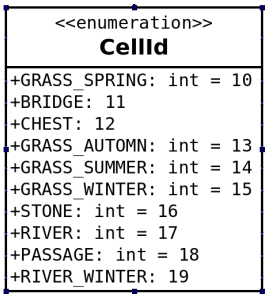


FIGURE 7 – Les différents type de CELL(case)

2.2 Conception Logiciel

Ci dessous vous trouverez le dia du state

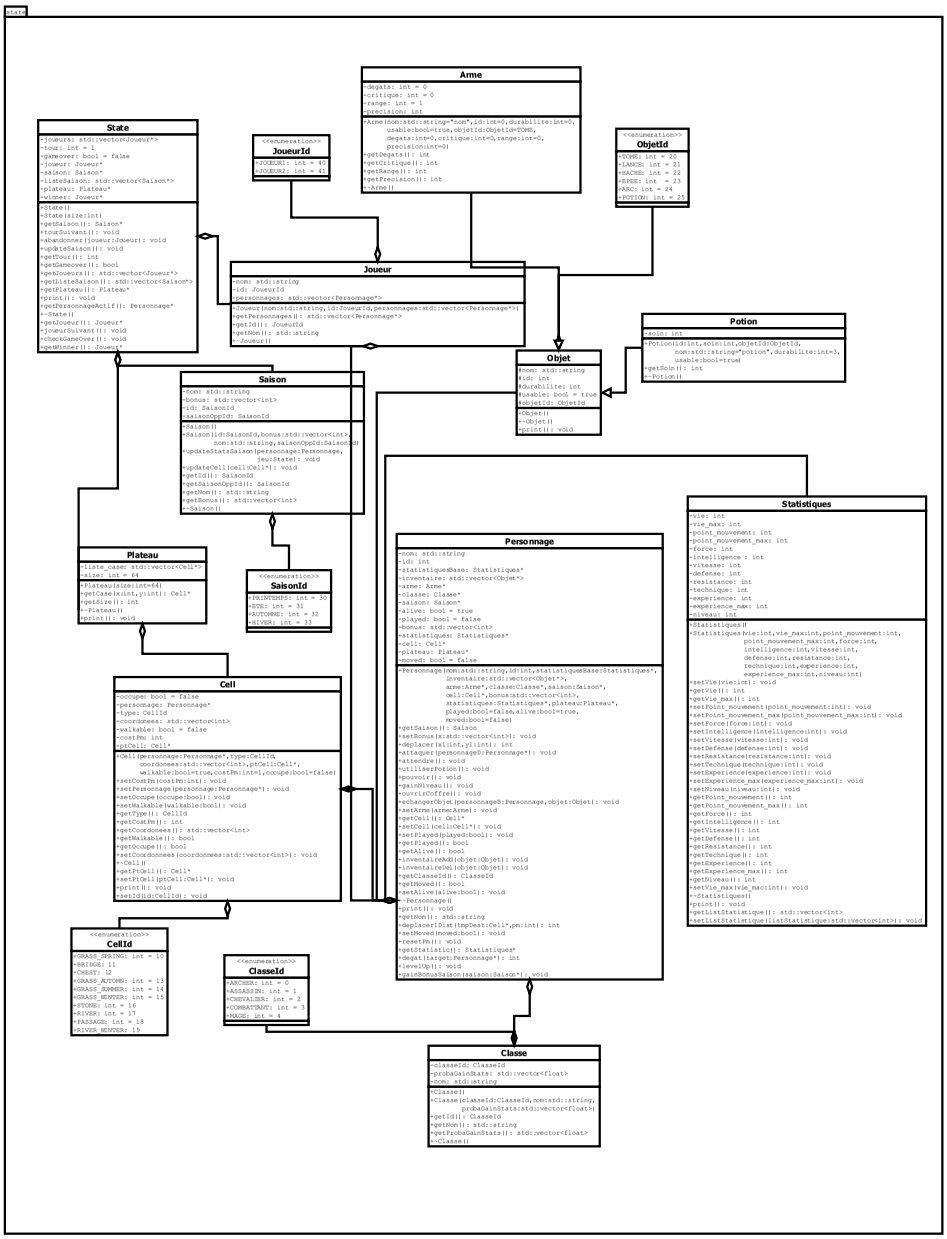


FIGURE 8 – Diagramme des classes d'état.

3 Rendu : Stratégie et Conception

3.1 Stratégie de rendu d'un état

On va utiliser une stratégie de rendu simple. On va procéder par couche. En effet, on peut facilement diviser notre jeu en 3 couches qui se superposent avec de bas en haut : le plateau, les personnages et l'interface. Plus la couche est basse, moins elle a besoin de s'actualiser fréquemment. En effet la couche Plateau doit s'actualiser une fois par tour, la couche personnage doit attendre des événements de type déplacement (le personnage bouge) ou attaque (si le personnage meurt). Tandis que la couche interface elle s'actualise pour chaque mouvement de la souris de l'utilisateur. Pour le moment notre render gère uniquement la couche plateau et la couche personnages.

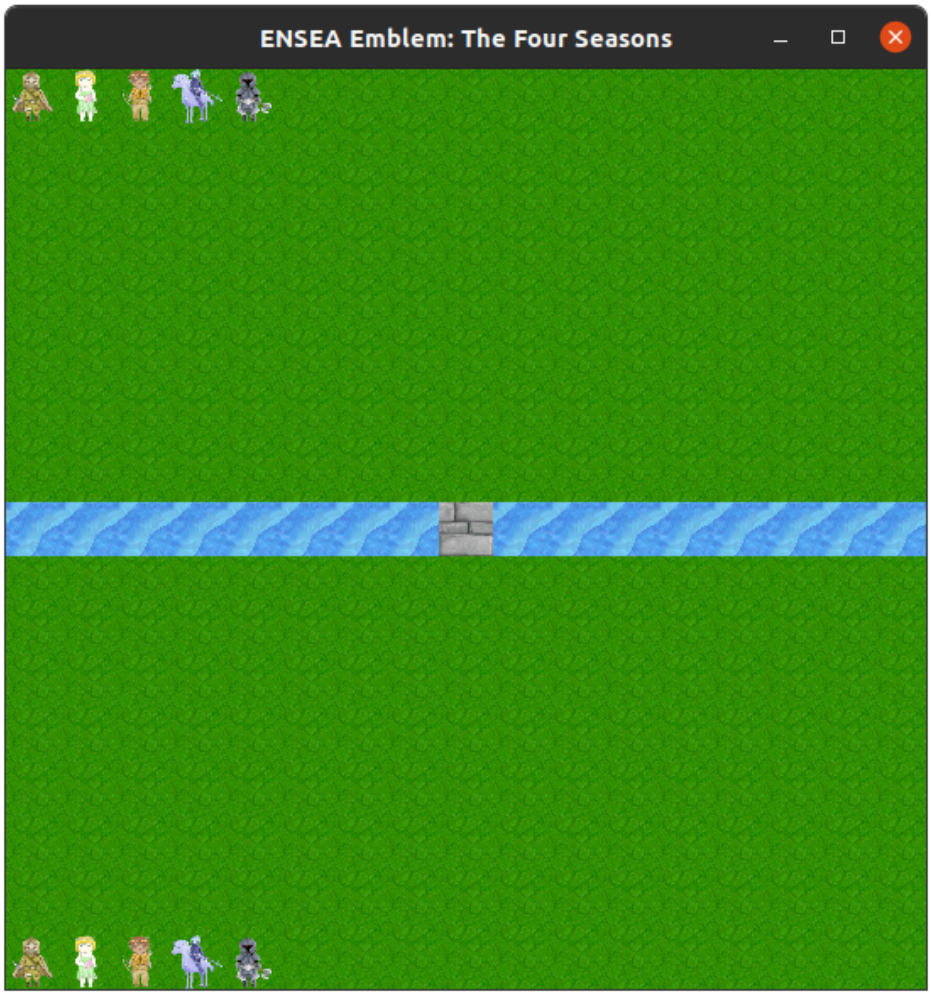


FIGURE 9 – le rendu pour 10 personnages sur un plateau de 17*17 cases

3.2 Conception logiciel

Le rendu est composé (pour le moment car on a implémenté seulement 2 couches) de 3 classes.

La première classe StateLayer a pour objectif de récupérer du state toutes les informations utiles dont on a besoin pour afficher le rendu. Par exemple, elle doit récupérer l'Id et la position de chaque case du plateau et l'Id de la classe et la position de tous les personnages. De plus, elle doit aussi savoir quel tileset est utilisé pour chaque couche (On utilise 2 tileset, un contenant toutes les cases et un contenant tous les personnages).

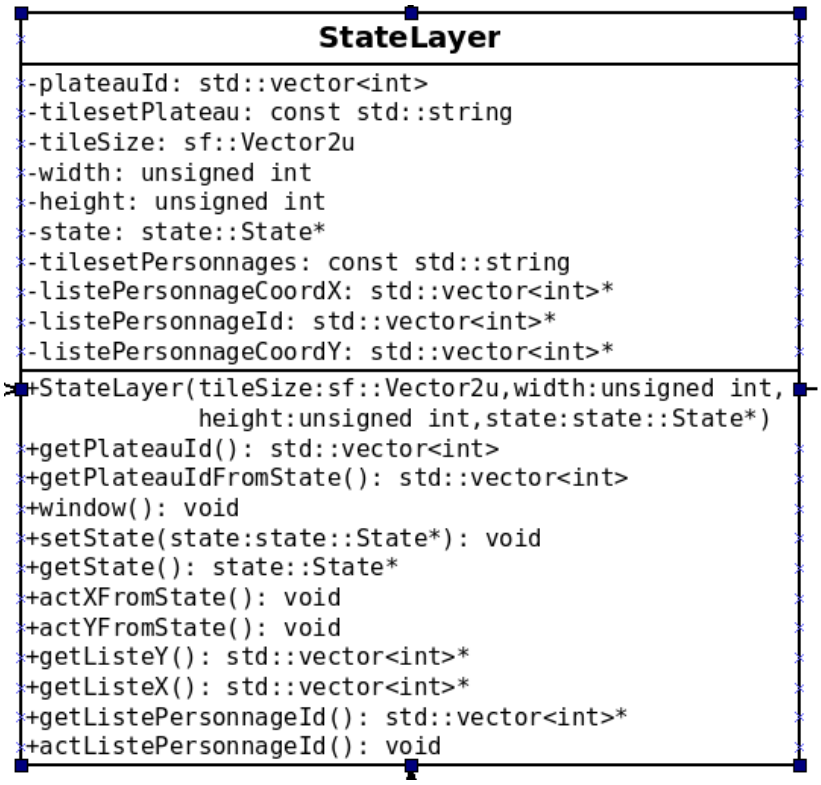


FIGURE 10 – La classe StateLayer

La classe surface elle doit utiliser toutes les informations que lui donne StateLayer pour afficher un rendu du plateau. Pour celà elle crée une liste de Vertex (dans notre cas ce sera des quads) et elle associe à chaque vertex une position dans la fenêtre du rendu et une position de texture dans le tileset. Elle à le même fonctionnement que celui décrit dans la documentation sfml (https ://www.sfml-dev.org/tutorials/2.3/graphics-vertex-array-fr.php).

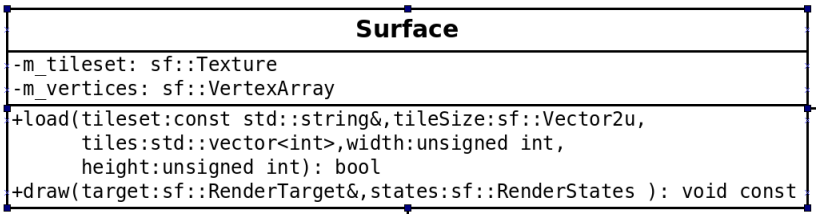


FIGURE 11 – La classe surface

La classe personnage est similaire à la classe surface.

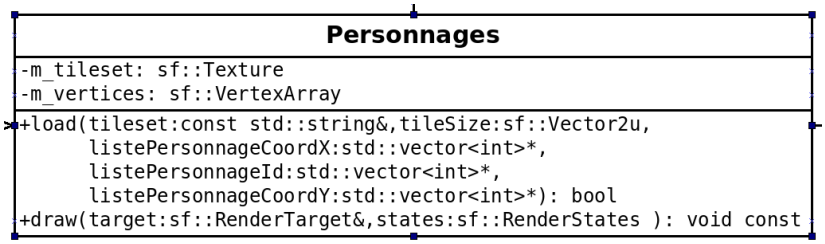


FIGURE 12 – La classe personnage

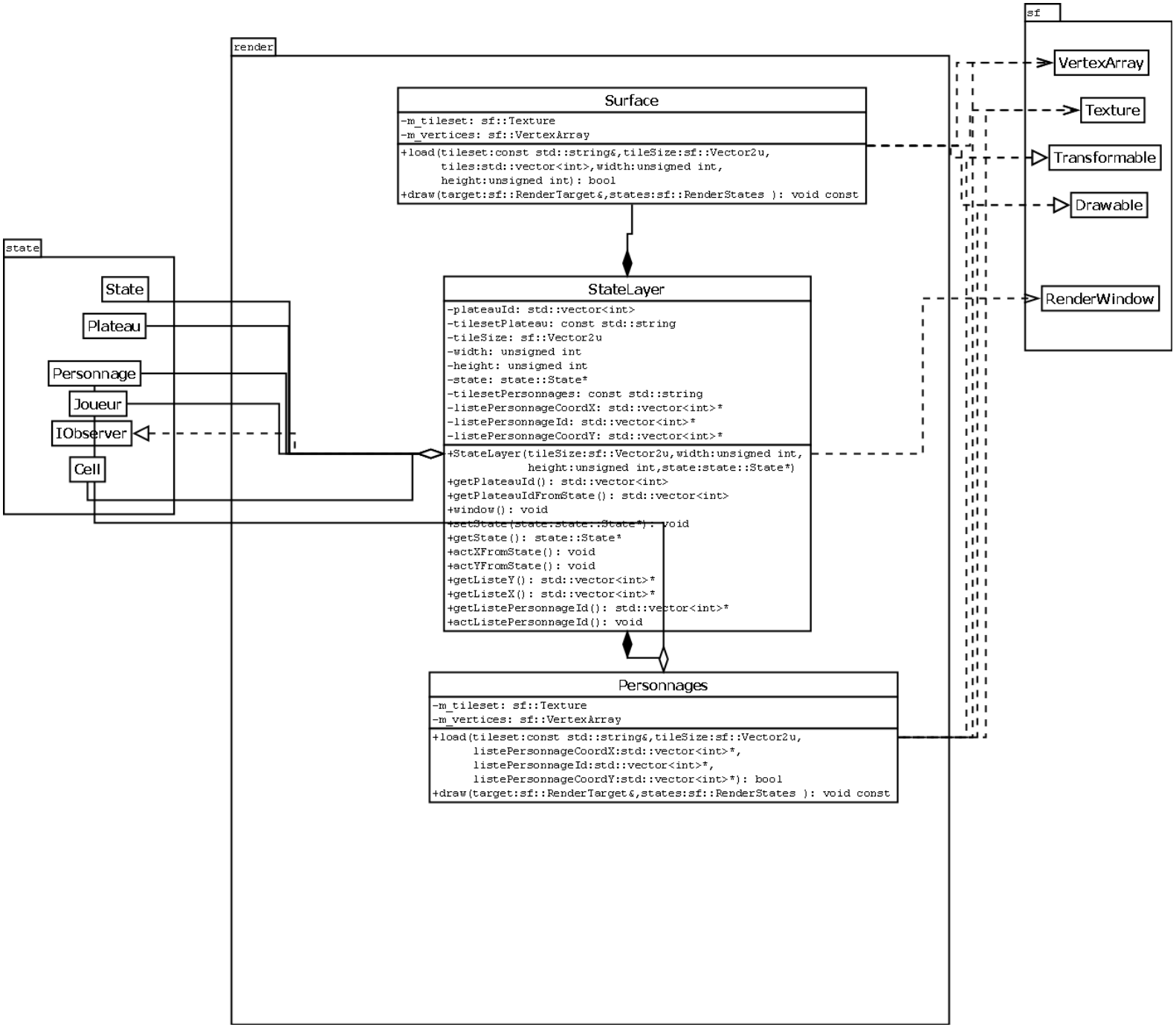


FIGURE 13 – Diagramme des classes de rendu.

4 Règles de changement d'états et moteur de jeu

4.1 Conception logiciel

Le diagramme des classes pour le moteur est présenté au dessous :

Le rôle des classes Commande est de lier toutes les commandes possibles dans le jeu. Pour chacune de ces classes, on défini un type de commande avec CommandId : .

- Commande Attaque
- Commande Déplacer
- Commande Échanger objet
- Commande Attendre
- Commande Arme
- Commande Utilisation potion
- Commande Fin tour
- Commande Ouvrir coffre

La commande attaque permet lancer une attaque et de choisir une case à cibler et d'attaquer le personnage sur cette case (si il y en a un). La commande déplacer permet de déplacer un personnage dans une direction choisi, si c'est possible.La commande échanger objet permet d'échanger des objets sélectionnées lorsque deux personnages sont proches.La commande attendre permet de faire attendre un personnage pour un tour. La commande arme permet de choisir une arme à équiper dans son inventaire. La commande utilisation potion permet au personnage de se soigner en buvant une potion, si il en a une dans son inventaire. La commande fin tour est pour faire attendre un tour tous les personnages du joueur qui active cette commande. La commande ouvrir un coffre permet à un personnage d'ouvrir un coffre si ce personnage est juste à côté de celui-ci.

Engine est le cœur du moteur, on utilise la méthode update pour exécuter les commmandes.

Dans la classe de CommandMove, l'attribut MoveId est lié avec la classe MoveId qui gère la direction. currentState ici est pour lier avec le state. Dans la méthode exécute de la classe CommandMove, on reçoit une direction du déplacement, et si l'endroit que le personnage active veut déplacer est bien dans le plateau, ce personnage bougera vers cette direction.

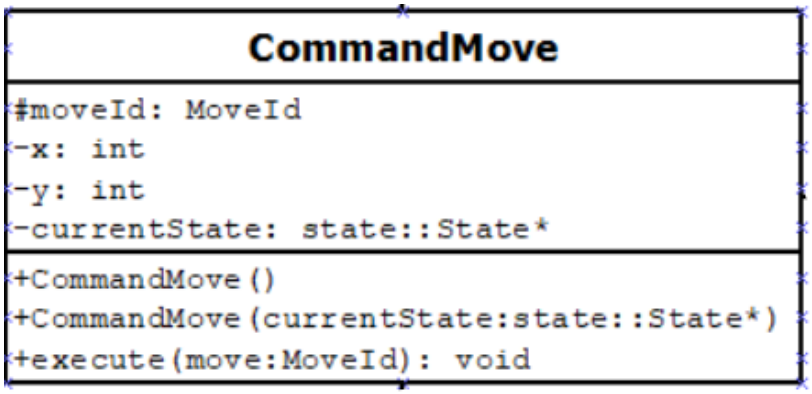


FIGURE 14 – La classe CommandMove

Dans la classe de CommandAtack, la méthode exécute permet de lancer la méthode attaquer dans le state, dans la classe l'engine, avant on exécute la commande attaque et lorsqu'on reçoit une direction d'attaquer,

il faut d'abord vérifier l'endroit choisie est bien dans le plateau et ensuite, on détecte s'il y a un ennemie à côté du personnage active et après on exécute la commande attaque.

Dans la classe de CommandAttendre, la méthode exécute permet de lancer la méthode attendre dans le state.

Voici un résumé du fonctionnement de l'engine :

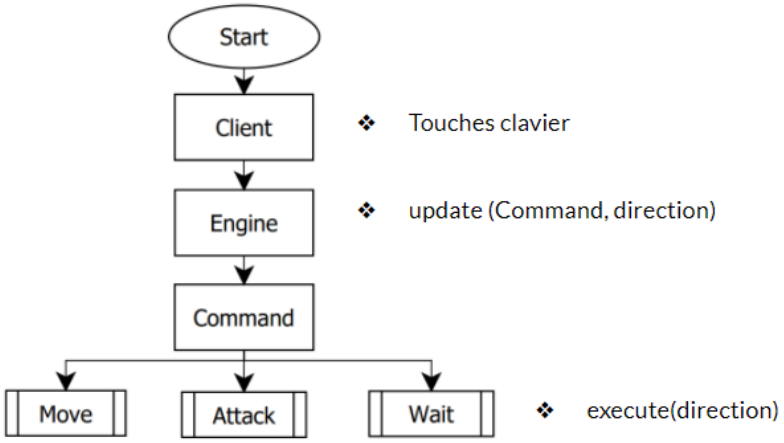


FIGURE 15 – Résumé du fonctionnement de l'engine

Lorsque on lance le jeu, le client initialise le jeu. Ensuite dès qu'il reçoit une touche clavier, il l'a traduit en commande et l'envoie à l'engine qui vérifie si la commande est faisable. Si oui il modifie le state.

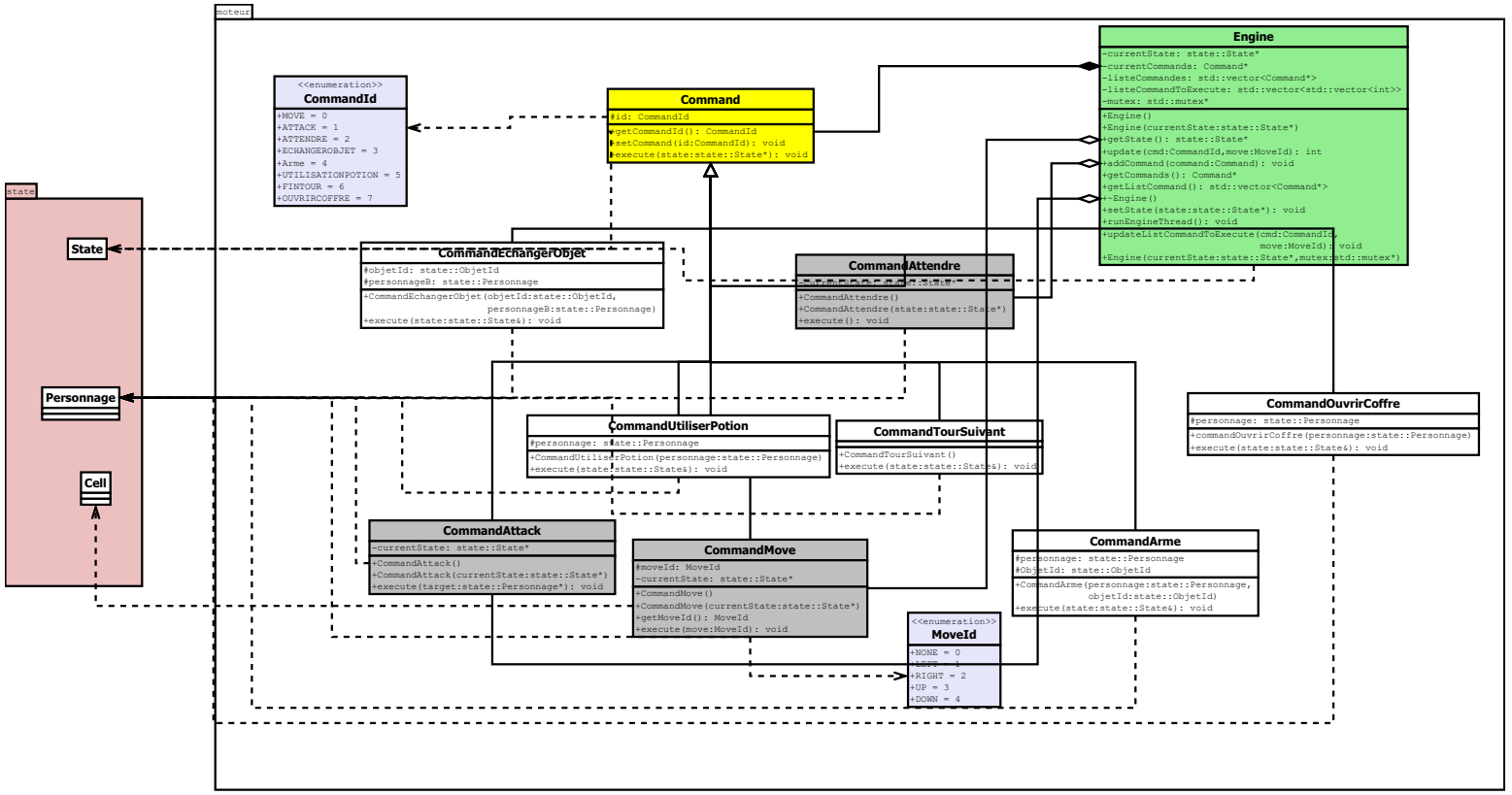


FIGURE 16 – Diagramme des classes d'état.

5 Intelligence Artificielle

On va maintenant implémenter des intelligences artificielles, chacune aura une stratégie très différente.

5.1 Stratégies

5.1.1 Intelligence aléatoire

L'intelligence artificielle aléatoire est la plus simple de toute, sa stratégie est basée sur le diagramme suivant :

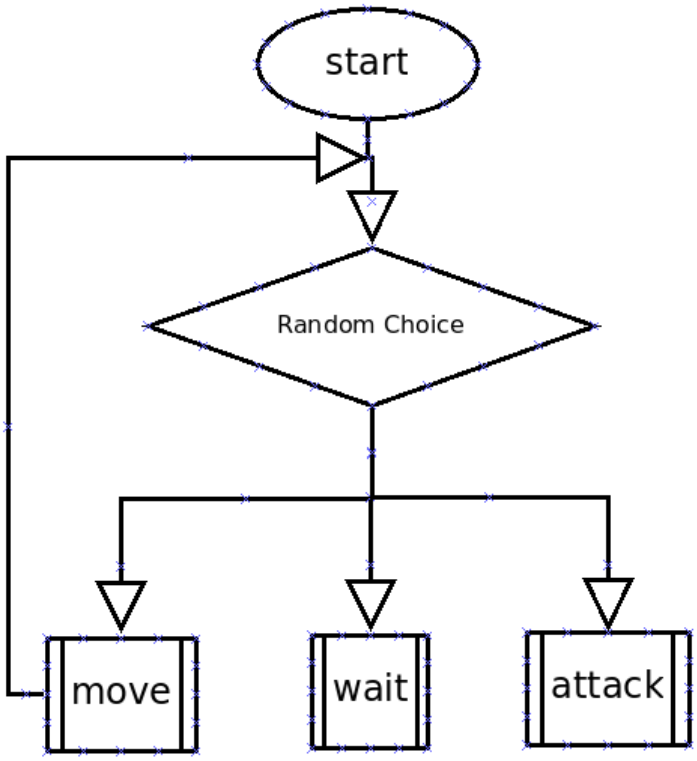


FIGURE 17 – La stratégie de l'ia aléatoire

Lorsque le client demande à l'ia de choisir une action à faire pour un personnage, l'ia va choisir aléatoirement si elle attaque, se déplace d'une case ou attend. Si elle choisit d'attaquer ou de bouger elle doit choisir une direction, elle va donc choisir aléatoirement 1 des 4 MoveId possible. Si la case à attaquer ou sur laquelle l'ia veut se déplacer n'est pas valide, l'engine va dire que c'est pas possible, ainsi on revient à la même situation qu'au départ et le client va à nouveau demander à l'ia de choisir aléatoirement une nouvelle commande. Si l'ia réussit à faire bouger un personnage sur une case valide, alors elle va devoir choisir une nouvelle commande pour ce personnage. Mais si elle choisit de faire attendre le personnage ou si elle réussit à le faire attaquer un personnage alors le personnage aura finit son tour et on va passer au personnage suivant.

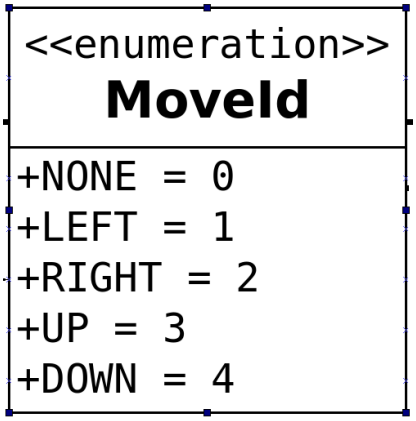


FIGURE 18 – La classe MoveId

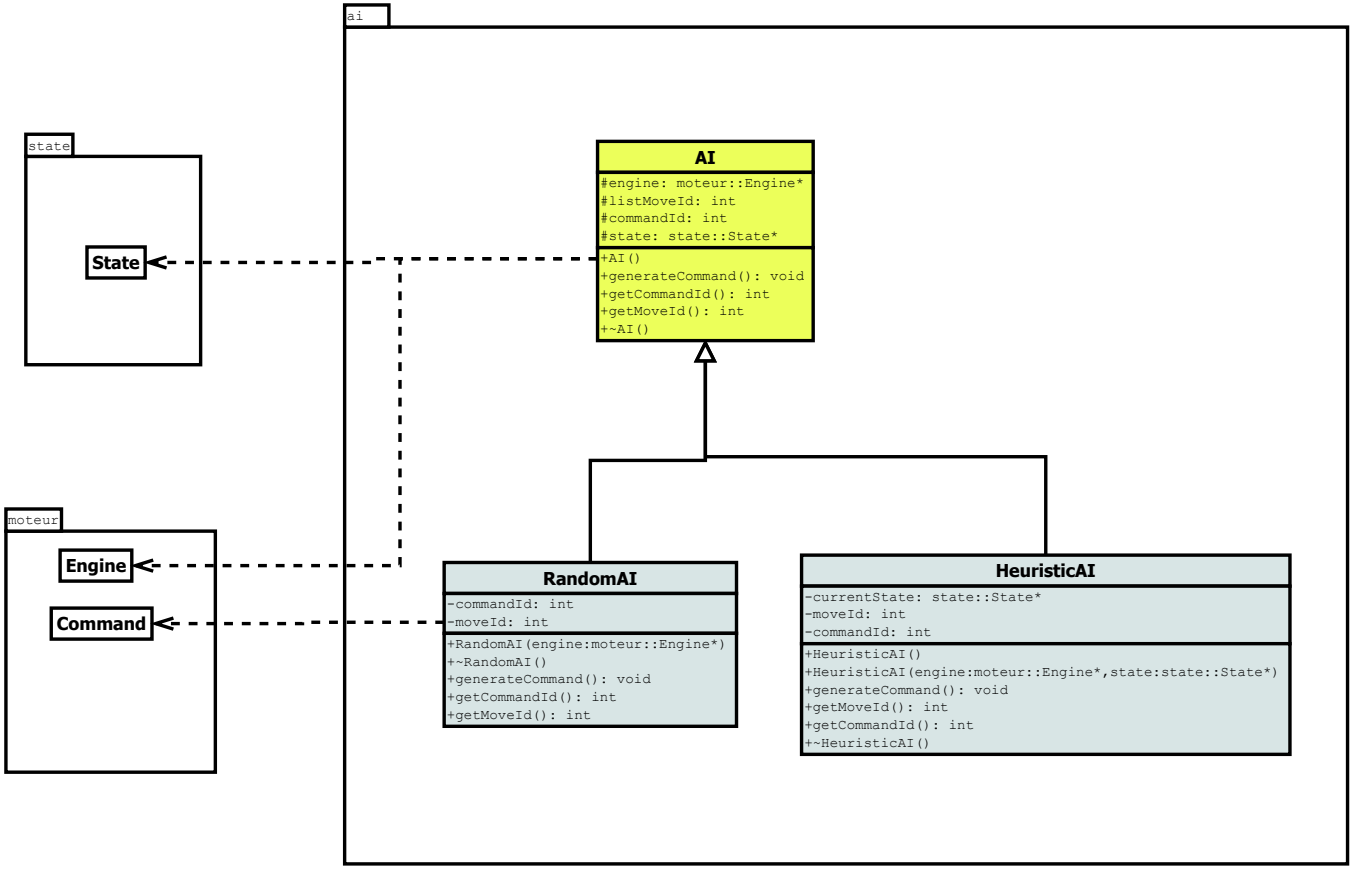
5.1.2 Intelligence Heuristique

Pour créer une intelligence artificielle un peu plus intelligente, on va rajouter des heuristiques. Ainsi cette intelligence artificielle va regarder quel est le personnage le plus près d'elle. Ensuite elle va se déplacer vers lui, si elle est à coté de ce personnage elle va l'attaquer mais si elle n'arrive pas à se rapprocher suffisamment proche pour l'attaquer car elle a plus de point de mouvement, elle va attendre. Si elle rencontre un obstacle sur le chemin qui est pour elle le plus court (case unwalkable ou occupée), alors elle va aussi attendre. Pour éviter que l'IA attende devant la rivière jusqu'à ce qu'elle gèle, on rajoute une heuristique qui fait en sorte que s'il y a une rivière entre elle est le personnage le plus proche, l'ia se dirige vers le pont le plus proche.

Pour tester les performances de l'IA heuristique par rapport à l'IA random. On leur fait jouer 100 parties l'une contre l'autre. L'IA heuristique gagne 91 parties, elle est donc beaucoup mieux. Les 9 parties gagnés par l'IA random sont probablement causés par l'aléatoire dans le système des dégâts (esquives).

5.2 Conception logiciel

On crée une classe IA abstraite et une classe pour chaque type d'ia. Chaque type d'ia hérite de la classe IA.


ai
state
State
AI
#engine: moteur::Engine*
#listMoveId: int
#commandId: int
#state: state::State*
+AI()
+generateCommand(): void
+getCommandId(): int
+getMoveId(): int
+~AI()
moteur
Engine
Command
RandomAI
-commandId: int
-moveId: int
+RandomAI(engine:moteur::Engine*)
+~RandomAI()
+generateCommand(): void
+getCommandId(): int
+getMoveId(): int
HeuristicAI
-currentState: state::State*
-moveId: int
-commandId: int
+HeuristicAI()
+HeuristicAI(engine:moteur::Engine*,state:state::State*)
+generateCommand(): void
+getMoveId(): int
+getCommandId(): int
+~HeuristicAI()

6 Modularisation

6.1 Organisation des modules

On va diviser le jeu en 2 threads, une première thread qui va éxécuter le Client et le Render et une deuxième thread qui va éxécuter l'engine.

Pour communiquer les threads vont utiliser une liste qui va contenir la liste des commandes (une commande est définie par une liste qui contient un CommandId et un MoveId) que l'engine doit exécuter. Cette liste est protégée par un mutex. Le thread engine exécute toujours le premier élément de la liste. Dès qu'il a exécuté la commande l'engine va supprimer cette commande de la liste.

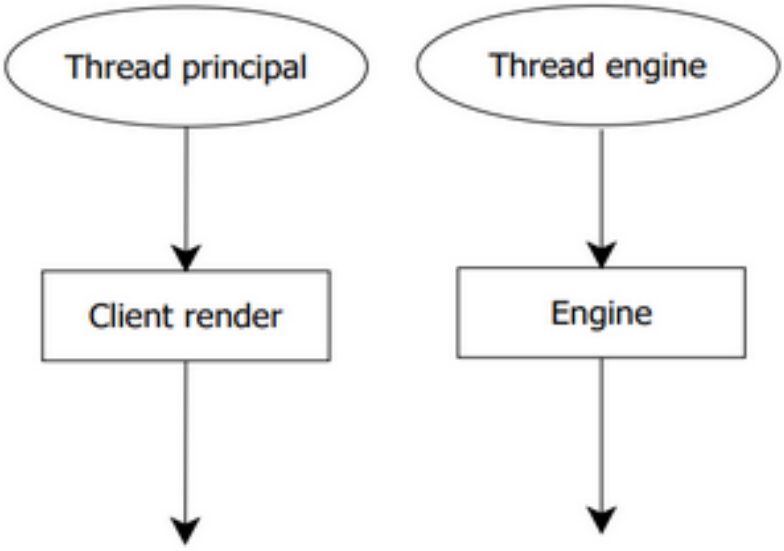


FIGURE 20 – Le schéma des threads

Ainsi la nouvelle classe engine adaptée au multithreading ressemble à ça

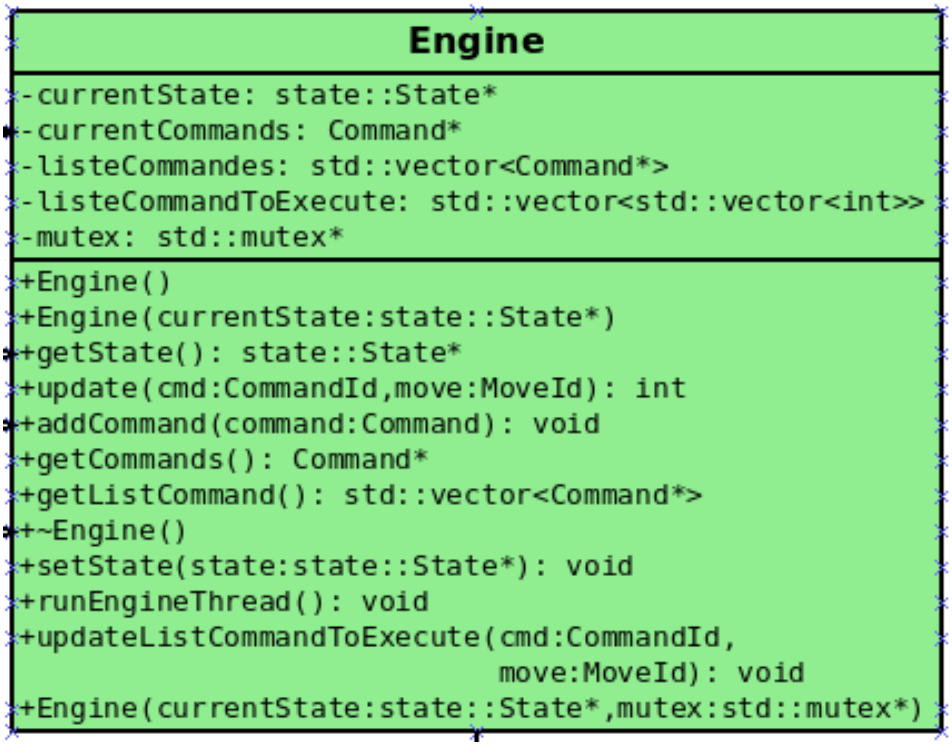


FIGURE 21 – La nouvelle classe engine