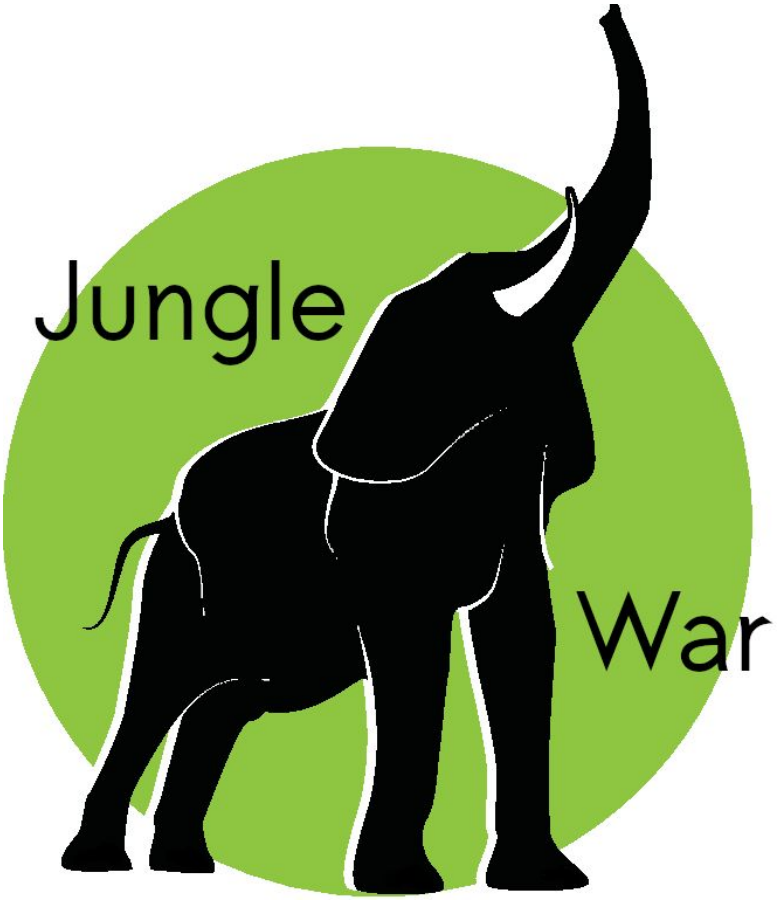
Rapport projet IS

Baptiste LANOUE et Robin SICSIC

25/09/2019 Jalon 1.1
16/10/2019 Jalon 1.final
31/10/2019 Jalon 2.1
6/11/2019 Jalon 2.2

Jungle War



Description du jeu

Archétype du jeu : Stratego

Stratego est un jeu stratégique où 2 joueurs s'affrontent et dont les mécanismes de jeu sont essentiellement le bluff et l'affrontement.
Le but du jeu est d'atteindre le drapeau adverse. Pour cet objectif, les joueurs disposent de 40 pièces qui ont des valeurs de puissances différentes et croissantes. Ces pièces sont face cachés. Pour découvrir la valeur d'une pièce, le combat est nécessaire.

Stratego est inspiré d'un ancien jeu chinois "Jeu du combat des animaux" qui est plus simple que le Stratego et que nous allons remettre au goût du jour.

Ce jeu se joue avec 8 pièces par joueur qui sont placées sur les carreaux. Les 2 camps sont le blanc (ou le rouge) et le noir (ou le bleu).

Chaque joueur dispose de huit animaux différents en tant que pièce de jeu. Les animaux ont leur propre valeur de puissance et certains ont des déplacements spécifiques.

Chaque pièce peut capturer une pièce adverse de rang égal ou inférieur. Cependant un rat (qui a la valeur la plus faible) peut capturer un éléphant (qui a la valeur la plus forte).

Fonctionnement du jeu

Deux joueurs s’affrontent sur un plateau avec des animaux. Il doivent prendre le contrôle de la tanière de l’adversaire ou capturer toutes les pièces de l’adversaire Chaque pièce a un score de puissance qui permet de manger les autres pièce inférieures ou égale et peut avoir des mouvements spéciaux. Les pièces ne sont pas face cachés.

Liste des pièces (classe Animal)

Ci-joint le tableau, des pièces du jeu avec la valeur de puissance et spécialité.

Pièce	Puissance	Spécialité
ELEPHANT	8	Perd contre le RAT uniquement si le RAT attaque par une case EARTH
TIGER	7	Si TIGER est sur une case SHORE, il peut se déplacer vers la case SHORE associé en face de sa case initiale.. Cependant si le RAT allié ou ennemis est sur sa trajectoire via une case WATER, TIGER ne peut pas se déplacer.
LION	6	Si LION est sur une case SHORE, il peut se déplacer vers la case SHORE associé en face de sa case initiale.. Cependant si le RAT allié ou ennemis est sur sa trajectoire via une case WATER, LION ne peut pas se déplacer.
LEOPARD	5	Si LEOPARD est sur une case SHORE, il peut se déplacer vers la case SHORE associé en face de sa case initiale.. Cependant si le RAT allié ou ennemis est sur sa trajectoire via une case WATER, LEOPARD ne peut pas se déplacer.
DOG	4	Pas de spécialité.
WOLF	3	Pas de spécialité.
CAT	2	Pas de spécialité.
RAT	1	Peut marcher dans les cases WATER Tue ELEPHANT uniquement si il se déplace depuis une case EARTH.

Priorité à l'attaquant : Si une pièce avance sur une case dont l'occupant est une pièce adverse de même valeur. C'est l'attaquant qui remporte.

Chaque joueur a un animal de chaque type, rangé par puissance croissante dans un dictionnaire de type "unordered_map".

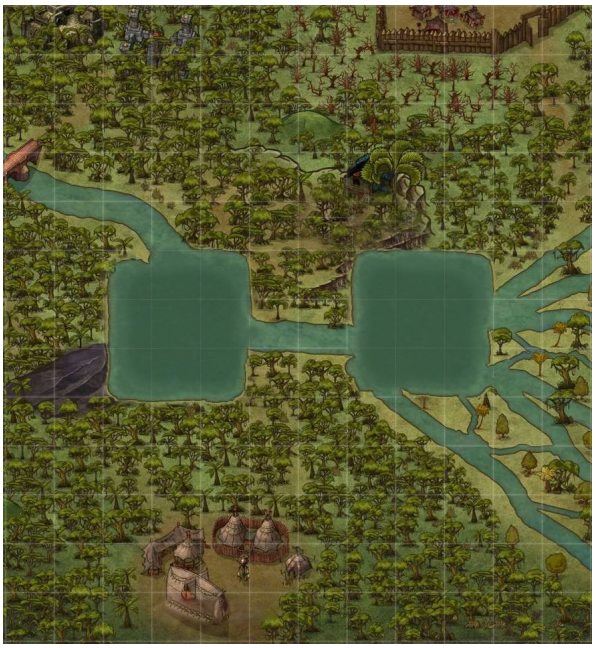
Liste des cases (classe Square)

Type de Case (énumération)	Spécialité
EARTH	Tout animal peut avancer sur une case EARTH.
WATER	Le RAT peut se déplacer dans les cases WATER. TIGER, LION, LEOPARD peuvent sauter horizontalement et verticalement haut dessus des cases WATER depuis une case SHORE si le RAT allié ou ennemis n'est pas sur la trajectoire.
SHORE	TIGER, LION, LEOPARD peuvent sauter horizontalement et verticalement haut dessus des cases WATER depuis une case SHORE si le RAT allié ou ennemis n'est pas sur la trajectoire.
TRAPJ1	Si un animal de J2 est sur la case TRAPJ1 de l'opposant , il passe dans l'état TRAPPED (voir description des états au jalon 1.final) , ce qui signifie que sa puissance est réduite à 0.
TRAPJ2	Si un animal de J1 est sur la case TRAPJ2 de l'opposant , l'animal passe dans l'état TRAPPED, ce qui signifie que sa puissance est réduite à 0.
THRONEJ1	Si un animal de J2 est sur la case THRONEJ1 de l'opposant, l'animal passe dans l'état VICTORIOUS et le joueur qui possède l'animal gagne la partie.
THRONEJ2	Si un animal de J1 est sur la case THRONEJ2 de l'opposant, l'animal passe dans l'état VICTORIOUS et le joueur qui possède l'animal gagne la partie.

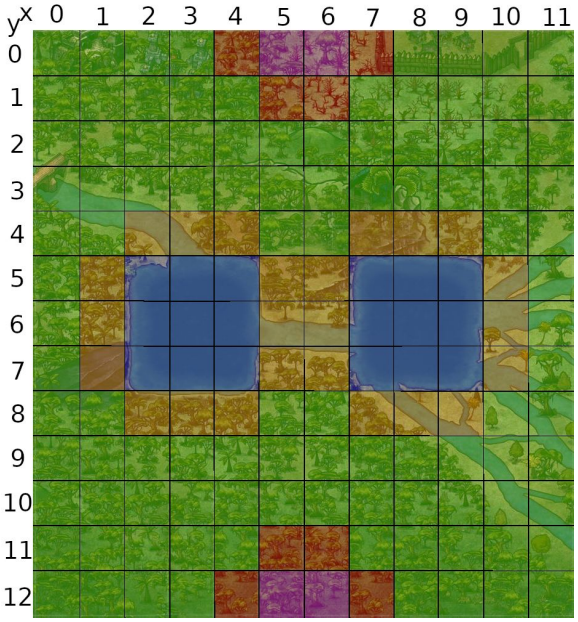
Ressources

Ressource du terrain : plateau de la jungle

D’après l’auteur, utilisation et modification libre.



Version utilisée pour la programmation avec les coordonnées :



Vert : EARTH
Orange : SHORE
Bleu : WATER
Rouge : TRAP (J1 ou J2)
Rose : THRONE (J1 ou J2)

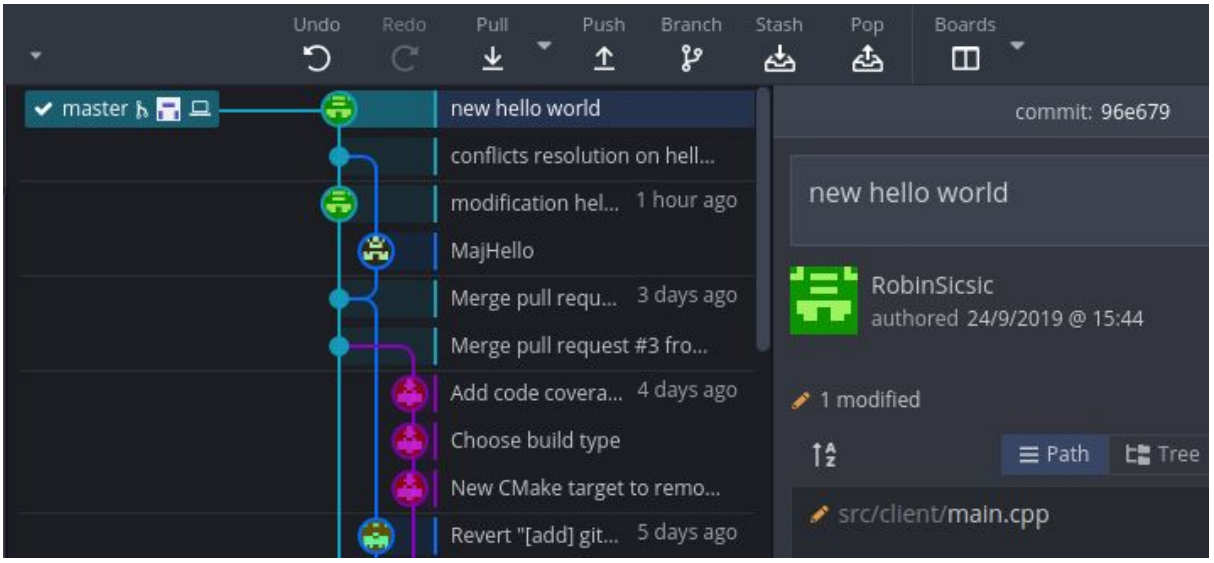
Ressources des pièces : animaux

Pour le choix des animaux, nous avions décidé de récupérer des création un peu partout sur internet. Nous les avons retravaillés pour qu'elle soient plus cohérentes en elles et dans le même thème. Elles sont en HD mais ne sont cependant pas libres de droits. Nous avons donc opté pour de nouvelles ressources, cette fois-ci libre de droit en utilisation et modification. Nous avons trouvé les ressources mais elle sont en cours de modification pour être adaptée au jeu.

Environnement de développement

Changement d'environnement depuis le dernier rapport. Nous sommes tous les deux passé sur un linux ubuntu 18.04 en dual boot.

Une clé SSH a été générée avec Gitkraken et enregistrée sur github. On peut ainsi administrer le git depuis ce logiciel et régler les conflits.
On peut voir ici quelques modifications (push) du programme main avec le "hello world".



Pour l'édition de code nous utilisons Atom et pour le diagramme de classe nous utilisons Dia.

Description et Conception des états

Description des classes

State

Un état du jeu (State) est formée par une grille (grid) de cases (Square) ainsi que deux joueurs (player1 et player2). L'attribut *turn* permet de savoir à quel tour de jeu en est la partie. La grille est un vecteur de vecteurs de cases (Square), il représente le terrain. Elle est ainsi initialisée par le constructeur State() et State(string nom1, string nom2) de la classe State. Elle ne sera pas modifiée au cours du jeu.

Player

Les joueurs possèdent eux même une liste d'animaux (animals), un nom, une couleur et une caractéristique booléenne d'être en train de jouer ou non.

La liste d'animaux de chaque joueur est un unordered_map qui permet d'associer un animal à un identifiant (de 1 à 8) représentant par ailleurs sa puissance et donc son type (chat, chien, etc.). L'utilisation d'un unordered_map permet de simplifier l'état en évitant une énumération d'animaux en plus d'un attribut "puissance" (information redondante).

Animal

Chaque animal a un statut variable (AnimalStatus) compris entre 1 et 5, ainsi que des coordonnées (x et y) sur le plateau.

Description des états des animaux:

Etat	Description
NORMAL	L'animal est dans son état normal, sur une case EARTH, vivant et de puissance non modifiée.
SWIMMING	L'animal est en train de nager, sur une case WATER, vivant et de puissance non modifiée. Il ne peut pas attaquer les autres animaux se trouvant sur les cases SHORES.
TRAPPED	L'animal est dans un piège ennemi, sur une case TRAPJ1 ou TRAPJ2 adverse, vivant mais puissance réduite à 0.
DEAD	L'animal est mort, n'est plus représenté sur le plateau de jeu mais toujours présent dans la liste des animaux du joueur.
VICTORIOUS	L'animal est sur une case THRONEJ1 ou THRONEJ2 adverse, le joueur associé à l'animal a immédiatement gagné.

Square

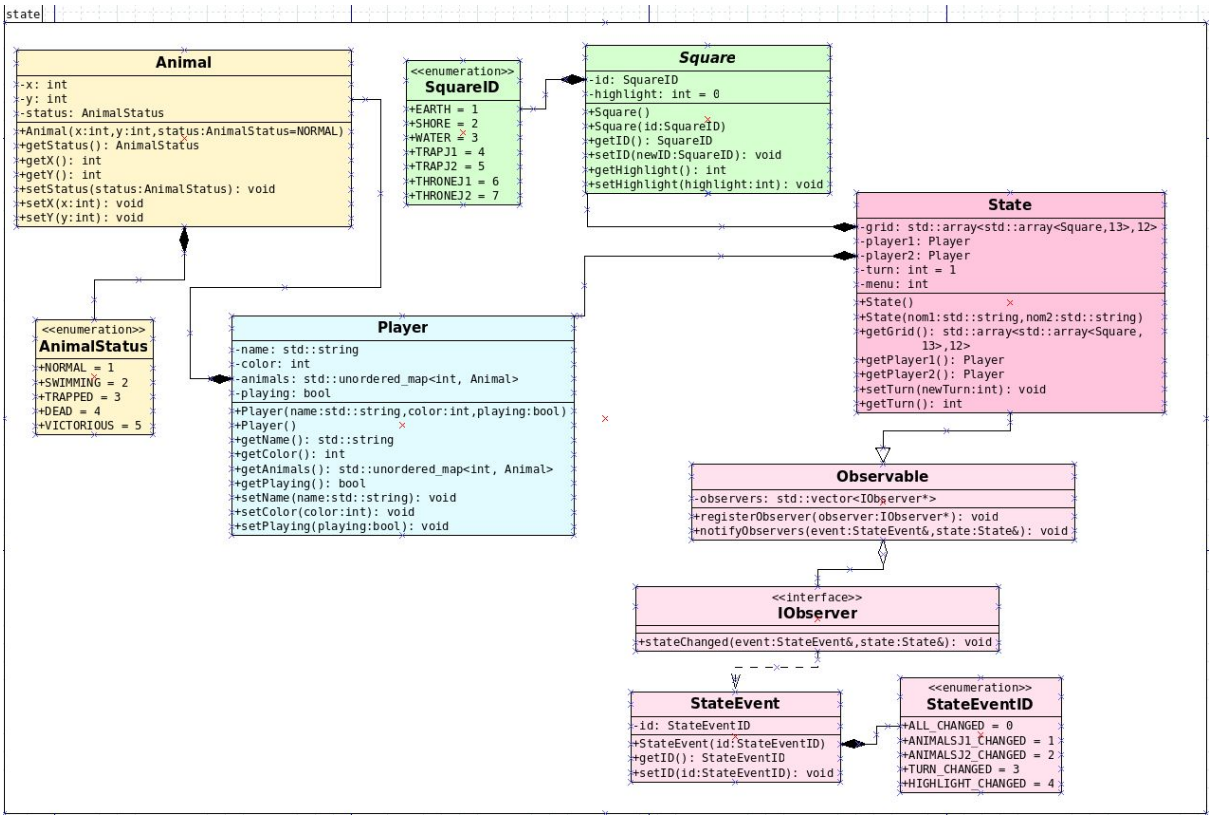
Chaque case du plateau a un identifiant “id” invariable (type SquareID) qui représente son type (EARTH, WATER, etc.) ainsi qu’une caractéristique booléenne d’être occupée ou non. On connaît les coordonnées d’une case de part sa position dans le vecteur “grid”.

Observeur

L’observateur de State notifie le rendu en cas de changement, avec un code d'événement pour que le rendu n’ai pas à tout redessiner, seulement ce qui a changé.
Les événements sont :

- ALL_CHANGED L’état doit totalement être redessiné.
- ANIMALSJ1_CHANGED Les animaux de J1 doivent être redessinés.
- ANIMALSJ2_CHANGED Les animaux de J2 doivent être redessinés.
- TURN_CHANGED Les informations liées au tour de jeu doivent être actualisées.
- HIGHLIGHT_CHANGED Les cases en surbrillances ont changées (surement suite à la sélection d’un animal) et doit être redessinées.

Diagramme de classe



Description et Conception du Rendu

Stratégie de rendu d'un état

Pour le rendu d'un état, nous avons fait le choix d'un rendu par tuile à l'aide de la bibliothèque SFML.

Nous divisons la scène à afficher en couches : une couche pour le terrain, une pour les jetons et une pour le descriptif des informations des joueurs. Nous rajouterons des couches afin par exemple d'afficher les cases sélectionnées.

La grille de terrain est créée à partir d'une seule image transformée en Sprite pour simplifier l'édition.

Les jetons sont des sprites qui sont placés sur la la grille du Terrain via les coordonnées initialisées dans le constructeur de la Player: *Player::Player(string name,int color,bool playing)* (src/shared/state/Player.cpp).

Ainsi lorsqu'on applique la commande ./bin/client renderTest2, la map est initialisée avec des jetons placés grâce à la méthode de RenderLayer *draw:(sf::RenderWindow& window)* dans notre main.cpp. Il prend un objet **state** et un objet **window**, la fenetre dans laquelle on veut afficher notre jeu.

Afin de tester l'initialisation, il est possible de modifier l'emplacement des jetons dans le constructeur de Player. Ce qui modifiera le rendu à l'écran.

Conception Logiciel

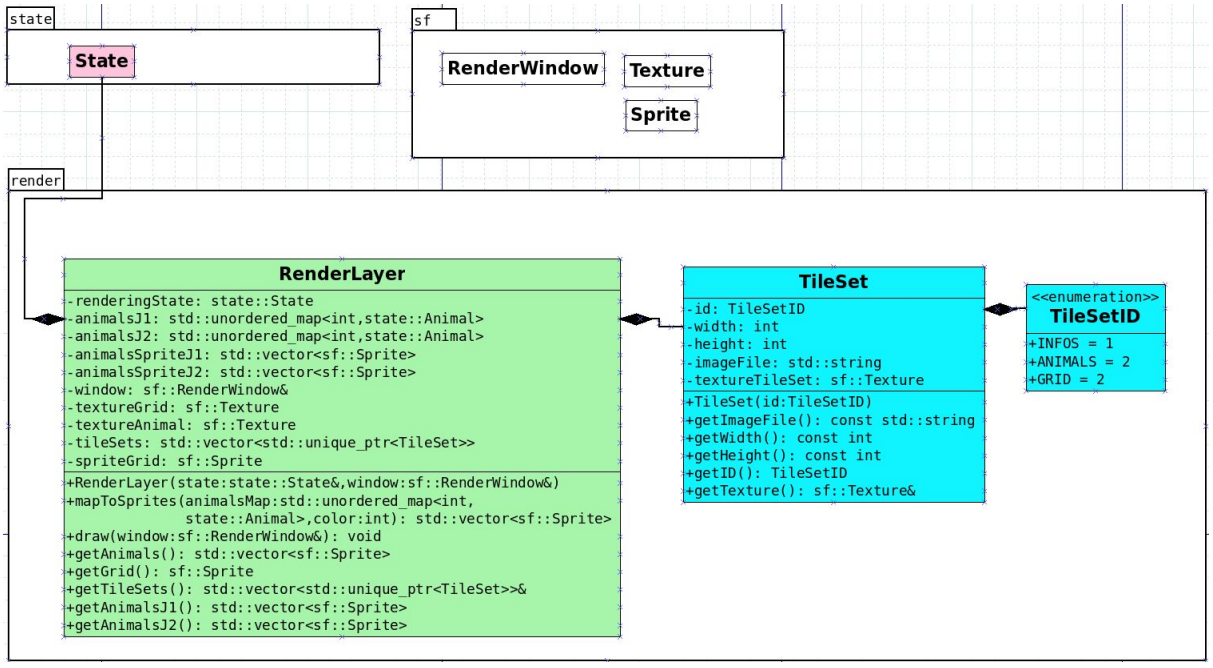
Classe RenderLayer : C'est la classe principale de notre rendu et permet l'affichage des différents éléments de l'état à afficher. Elle devrait être une classe observateur lié à la classe State. Elle possède comme attribut l'état du Jeu **renderingState,** les animaux des deux joueurs et leur sprites : **animalsJ1, animalsJ2, animalsSpriteJ1** et **animalsSpriteJ2,** la Window **window.** Les textures du terrain et des animaux respectivement **textureGrid** et **textureAnimals.** Un vecteur de pointeur TileSet **tileSets** qui sera utilisé plus tard car pour le moment, les images d'animaux peuvent être chargée directement en utilisant un seul fichier animalsTile.png **.**
Pour initialiser, la map on utilise *Draw:(sf::RenderWindow& window).*
Pour obtenir nos sprites à afficher à partir de la liste des animaux des joueurs, on utilise la méthode *mapToSprites(unordered_map<int,Animal> animalsMap, int color).* Elle modifie l'apparence l'emplacement des sprites en fonction du statut des animaux et de leur coordonnées.

Classe TileSet : Cette classe possède plusieurs attributs : un id de type **TileSetID**, des entiers **Width** et **Height** (dimension de en pixel d'une tuile) et une chaîne de caractères

imageFile (chemin vers un « *fichier.png* »). L'ID peut prendre différentes valeurs : INFO, ANIMALS ou GRID.

Suivant l'ID passé en argument du constructeur de TileSet, les attributs **Width**, **Height** et **imageFile** sont initialisés différemment. De plus, les textures sont chargées depuis le fichier correspondant lors de l'appel au constructeur, et donc une seule fois par **TileSet**. Les méthodes **getWidth, getHeight, getImageFile** et **getTexture** permettent de récupérer ces attributs. Pour le moment, la même sprite est utilisée pour tous les animaux mais nous allons utilisée **TileSet** pour avoir des sprites différentes.

Diagramme de classes de rendu



Règle de changement d’états et moteur de jeu

Suite à un événement provenant du rendu tel qu’un clic de souris, le moteur de jeu interprète la commande et effectue une modification de state. Une fois le state modifié, le rendu est notifié et l’affichage est actualisé.

En début de tour, le joueur peut sélectionner une pièce et puis la déplacer. La pièce peut se déplacer en fonction des règles.

Un animal ne peut être déplacé que d’une case par une case, en fonction de ses caractéristiques.

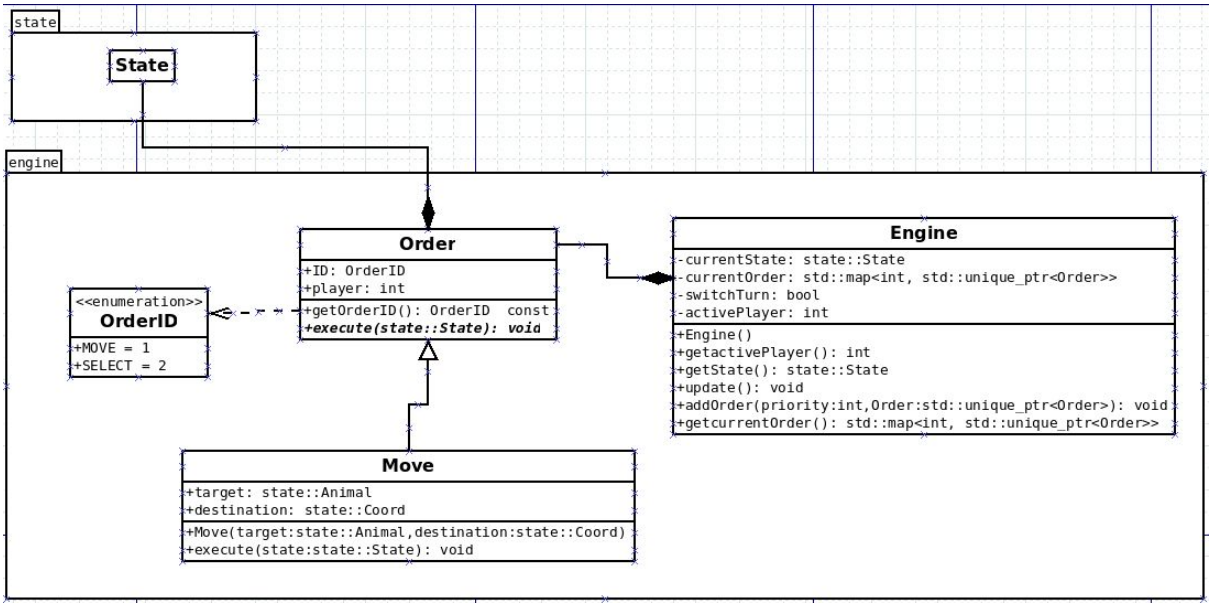
Un personnage ne peut attaquer un autre personnage que lorsque celui ci respecte les contraintes cités dans la description des règles et appartient au camp adverse (les attaques alliées ne sont pas autorisées).

Le tour de jeu d’un joueur est terminé lorsque qu’il a effectué un déplacement ou une attaque. C’est alors le tour du joueur adverse.

Lorsqu’un animal est attaqué par un ennemi, si l’animal a une valeur plus petite ou égale, il est passé dans le statut DEAD.

Si tous les animaux d’un joueur meurent, la partie est terminée et le joueur adverse gagne.

Diagramme de classe du moteur de jeu



Le diagramme des classes pour le moteur de jeu (« ***engine.dia*** ») est présenté ci-dessus. Le moteur de jeu repose sur le Design Pattern Command.

Classe Engine :. Elle permet de stocker les commandes dans une std::map avec clef entière (avec « addOrder »). Ce mode de stockage permet d'introduire une notion de priorité en anticipation à la version du jeu en lien avec un serveur : on traite les commandes dans l'ordre de leur clef (de la plus petite à la plus grande). Lorsque la méthode « update » est appelée, le moteur appelle la méthode « execute » de chaque commande puis supprime toutes les commandes (Order) une fois exécutées.

Classe Order : La classe Move, héritant de la classe Order, possèdent chacune une méthode « execute » qui fait effectuer à la pièce un déplacement.

Sources

Description stratego :
https://fr.wikipedia.org/wiki/Stratego

Description jeu des animaux :
https://fr.wikipedia.org/wiki/Jeu_du_combat_des_animaux

Elephant logo libre de droits :
https://publicdomainvectors.org/en/free-clipart/Elephant-silhouette-clip-art/79178.html

Plateau libre de droits :
https://www.drivethrurpg.com/product/273176/Jungle-Delta-Jungle-Map