Ter M1 : Algorithme min-max dans les jeux de stratégie historiques

Cahier des charges

Tuteur : Frédéric Koriche

Groupe :
Anthony Willemot
Quentin Bresson
Cédric Ogive
Sébastien Long

1/ Étude du besoin

Après une discussion avec notre encadrant, nous avons défini le but du projet comme suit :

Créer un jeu de stratégie historique (combat entre deux armées au tour par tour) et y implémenter une intelligence artificielle pouvant être de type min-max. Le choix de l'époque, des unités et des règles de jeu est laissé libre.

Ce projet sera effectué dans le cadre du TER du deuxième semestre de Master informatique. La date butoir est fixée au 30 avril pour le rapport final, aux 14 et 15 mai pour la soutenance.

2/ Fonctionnalités du programme

Il s'agit d'un jeu de stratégie au tour par tour, dans lequel deux armées s'affrontent sur une carte prédéfinie qui représente un terrain de combat. Le cadre est choisi est de type médiéval-fantastique classique.

L'utilisateur devra pouvoir :

- jouer contre un autre joueur humain (sur la même machine)
- jouer contre l'intelligence artificielle
- assister à un match entre deux intelligences artificielles

L'intelligence artificielle se verra dotée de trois niveaux de difficulté : facile, moyen et difficile, réglable dans le menu principal avant le lancement d'une partie (la difficulté ne pourra pas être changée en cours de partie). Elle déplace ses unités et ordonne ses attaques de la même façon qu'un joueur; il est donc possible de suivre son tour de jeu pour essayer de comprendre sa stratégie.

L'utilisateur aura plusieurs scénarios à sa disposition et pourra en changer à chaque nouvelle partie. Un scénario est défini par une carte représentant un terrain de jeu et un certain nombre d'unités données pour les deux armées.

Un système de sauvegarde sera aussi mis à disposition afin de pouvoir à tout moment enregistrer la partie en cours ou reprendre une partie préalablement enregistrée.

En cours de partie, l'utilisateur notamment aura la possibilité de :

- sauvegarder sa progression,
- charger une sauvegarde,
- recommencer le scénario du début,
- abandonner la partie.

En fin de partie un tableau de statistiques sera affiché regroupant quelques informations sur la partie qui vient de s'achever (nombre d'unités perdues, nombre de tours de jeu, etc...)

Aucun éditeur de terrain/unités n'est prévu (ceci afin de rester dans un délai de temps raisonnable), mais l'utilisateur pourra quand même modifier certains réglages ou ajouter un scénario via l'édition des fichiers de configuration.

3/ Règles de jeu

But du jeu :

Tuer le (ou les) capitaine(s) ennemi(s).

La carte :

Le plateau de jeu est composé d'une matrice de cases hexagonales. Chaque case sera affectée d'un type de terrain. Les différents types de terrain sont :

- Plaine : le terrain « de base ». Le déplacement et les attaques s'y déroulent normalement.
- Forêt : Les unités terrestres s'y déplacent plus lentement. Les archers savent utiliser cet environnement à leur avantage, et ils gagnent donc un léger bonus de défense, en raison de la couverture naturelle qu'offrent les arbres.
- Marais : Toutes les unités terrestres s'y déplacent plus lentement et y subissent un léger malus de défense. C'est encore plus prononcé avec les unités lourdes.
- Montagne : Toutes les unités terrestres s'y déplacent très lentement, car c'est le terrain le plus difficile à franchir. Il offre en revanche une défense supplémentaire. Les archers profitent de la hauteur des montagnes pour infliger plus de dégâts sur une plus grande portée.
- Fort : Les unités terrestres s'y déplacent normalement. C'est le type de terrain qui offre la meilleure défense : il est très difficile de déloger une armée réfugiée derrière des remparts. Toute unité terrestre se trouvant dans un fort dispose d'un bonus de défense et d'un malus en attaque.
- Eau peu profonde : les unités s'y déplacent un peu plus lentement qu'en plaine. Certaines unités lourdes ne peuvent pas s'y déplacer du tout. L'eau est un terrain à découvert, aussi les unités qui s'y trouvent subissent un malus de défense.
- Eau profonde : infranchissable, sauf par les unités volantes.

Caractéristiques des unités :

Chaque unité possède les caractéristiques suivantes :

- **HP :** Points de vies de l'unité
- **ATQ :** la force d'attaque de l'unité
- **DEF :** le niveau de défense de l'unité
- **Points de déplacement :** la vitesse de déplacement de l'unité.
- **Portée :** la portée maximale des attaques de l'unité.

Les unités ne se déplaceront pas seules mais en groupe. Sur une case donnée, on aura alors un groupe constitué de X unités de type Y. Un groupe est forcément constitué d'unités du même type. Lors des combats, le nombre d'unités et les statistiques du type d'unité seront pris en compte pour déterminer le résultat de la bataille.

Descriptif des unités :

Infanterie légère : Unité de base, elle n'est pas très puissante. Le fait qu'elle n'ait pas de forces ou de faiblesses particulières en fait une unité polyvalente et passe-partout.

Infanterie lourde : La force brute. Cette unité possède une très grande force de frappe et une défense robuste. Sa seule faiblesse est de se déplacer très lentement ce qui la rend particulièrement vulnérable contre des unités à distance, et tout particulièrement en terrain difficile.

Lancier : Cette unité n'est pas très puissante. Sa longue lance la spécialise contre la cavalerie. Lancer une charge de cavalerie contre une rangée de lancier équivaut très probablement à perdre sa cavalerie.

Cavalerie légère : Cette unité montée sur cheval est très rapide. Sa rapidité lui permet de contourner rapidement les lignes adverses, de charger les unités à distances ennemies ou de prendre rapidement des points stratégique. Elle est malheureusement un peu faible, et ne tiendra pas longtemps contre des unités lourdes.

Cavalerie lourde : La cavalerie lourde essaye de rattraper les faiblesses de la cavalerie légère en augmentant son armement et son armure. Pour cela elle sacrifie un peu de sa capacité de déplacement.

Archer : Équipé de son arc, l'archer peut attaquer les ennemis à distance. Placé sur une hauteur, il devient redoutable. Mais il faut faire attention à le couvrir ou faire en sorte que l'ennemi n'arrive jamais au corps à corps, car les archers se font décimer en quelques coups.

Griffon : Grand et puissant animal, le griffon inspire la peur aux ennemis. Une grande force d'attaque, une bonne défense et la capacité de voler font de cette unité un atout majeur dans une armée.

Dragon : Comme les griffons, les dragons ont la capacité de voler, ce qui annule toute pénalité due au terrain. Le dragon est puissant et son souffle peut faucher les ennemis à distance. Pour attaquer un dragon, il est préférable de réunir une bonne armée.

Capitaine : Il est puissant, mais seul, et doit être gardé en vie. S'il est tué, la partie est perdue.

Tour de jeu :

Le joueur ayant la main effectue une, plusieurs ou toutes les opérations qui lui sont permises, parmi lesquelles :

1 - Déplacement d'un groupe : effectué dès que le joueur en donne l'ordre. Les déplacements sont soumis à plusieurs contraintes :

a) Respect de la capacité de déplacement de l'unité.

b) Pour contourner un groupe ennemi, le groupe du joueur devra se tenir à au moins deux cases de distance. Il est possible de se déplacer à moins de deux cases d'un groupe ennemi mais alors ce déplacement est forcément final. Il est donc impossible de passer près d'un ennemi et de

poursuivre sa route sans l'affronter. Cela permet une gestion simple des fronts de combat.

2 - Scission d'un groupe : effectué dès que le joueur en donne l'ordre, et limité à une scission par groupe. Autrement dit, si un groupe se scinde en deux, les deux groupes engendrés ne pourront plus se scinder ce tour-ci (mais ils pourront chacun se diviser à nouveau au tour suivant). Les groupes se scindent forcément en deux groupes de taille identique (à une unité près, si le nombre d'unités du groupe d'origine était impair).

3 - Fusion de deux groupes : ils doivent être composés du même type d'unité, et fonctionnent de la même manière que les scissions. Autrement dit, un groupe déjà issu d'une fusion ce tour-ci ne peut pas se fusionner à nouveau.

4 - Attaque : le joueur indique les cibles ennemies à attaquer, mais les attaques ne sont effectuées qu'à la fin du tour, après tout le reste. Deux groupes d'unités peuvent attaquer le même groupe ennemi.

Lorsque le joueur a fini de donner ses ordres, il appuie sur un bouton pour finir son tour. Cela entraîne le déclenchement de toutes les batailles demandées.

Déroulement des batailles :

Lorsqu'une bataille est déclenchée, le programme calcule le résultat de la bataille et l'affiche. Chaque tour de jeu représente environ une heure de combat, et il peut donc y avoir des survivants dans les deux camps à la fin d'un tour.

4/ Choix de développement :

Outils :

Pour ce projet, nous avons opté pour le langage C++, principalement dans le but de nous préparer à ce que seront nos futurs emplois. L'interface du logiciel sera développée en 2D, à l'aide de la librairie SFML (Simple and Fast Multimedia Library). Il s'agit d'une alternative à SDL, aussi facile d'utilisation, mais avec des performances de loin supérieures. Pour la partie artistique, nous nous appuierons sur les images du jeu libre « Battle for Wesnoth ». Cela nous permettra de nous concentrer avant tout sur le principal : la mise au point de l'intelligence artificielle.

Architecture :

L'idée de base est de séparer le code relatif aux deux principaux types de responsabilités :

- la gestion du jeu : définir ce qui est permis, ce qui ne l'est pas, ce qu'il se passe lorsqu'une armée en attaque une autre, l'IA, etc.
- Les interactions avec l'utilisateur : il s'agit d'interpréter les actions de l'utilisateur d'une part, et de l'autre d'afficher l'état de la partie.

Le premier aspect est traité par le moteur de jeu, et le second par le moteur graphique. Ces deux composants manipulent les données de l'application, accessibles via un singleton. Afin de réduire au maximum leur couplage, les communications entre eux se font selon le principe du design pattern Observateur :

- La classe abstraite Moteur est à la base de tout objet observable. Mais puisque les moteurs

s'observent entre eux, ils jouent également le rôle d'observateur.
- Le moteur de jeu et le moteur graphique sont représentés par des classes dérivant de Moteur.
- Les appels de méthode d'un moteur sur l'autre se font en passant un objet Event en paramètre. Ces objets permettent d'identifier le type d'évènement dont il est question, et encapsulent diverses données au besoin (le chemin que prend une unité pour aller d'un point à un autre par exemple).

Revenons sur les objets Event. Ils sont à la base des « discussions » entre les moteurs, qui pourraient, s'ils étaient humains, ressembler à cela :
(on note «GUI » le moteur graphique, et « Jeu » le moteur de jeu)

GUI à Jeu : « Le joueur a sélectionné le groupe situé en (12,3) »
Jeu à GUI : « Voici les possibilités de déplacement pour ce groupe : (11,3),(12,4)(11,4) »
GUI à Jeu: « Le joueur souhaite que le groupe en (12,3) se déplace en (12,5) »
Jeu à GUI : « Ce déplacement est interdit »
GUI à Jeu : « Le joueur souhaite que le groupe en (12,3) attaque celui situé en (12,2) »
Jeu à GUI : « Permission accordée. L'attaque se fera à la fin du tour »
GUI à Jeu : « Le joueur a fini son tour (il souhaite donc lancer les attaques en attente) »
Jeu à GUI : « Résultat de la bataille : 142 mort chez vous, 112 chez l'autre »
Jeu à GUI : « Le tour est fini »
etc...

L'élaboration du diagramme de classes UML s'est faite en plusieurs étapes :
Définition de la structure de base du programme, indépendante des règles du jeu : les moteurs, le système d'évènement.
- Définition des structures de données.
- Définition des classes auxquelles les moteurs délèguent leur travail.

Le diagramme présent en annexe comprend les deux premières étapes, la troisième dépendant d'algorithmes qu'il nous reste à définir. Il est à noter qu'il n'est pas le reflet de l'implémentation finale : les setters et getters n'ont pas été ajoutés par souci de lisibilité, tout comme la portée des opérations et attributs, et les paramètres des opérations ont aussi été occultés.

5/ Organisation des tâches et planning

<table>
<tr><td>Semaine du</td><td>26/01/09</td><td>02/02/09</td><td>09/02/09</td><td>16/02/09</td><td>23/02/09</td><td>02/03/09</td><td>09/03/09</td><td>16/03/09</td></tr>
<tr><td>Anthony Willemot</td><td colspan="2" rowspan="2">Élaboration du gameplay, choix de conception, répartition des tâches, création d'un cahier des charges, etc.</td><td>Familiarisation avec SFML</td><td>Réalisation d'une interface graphique basique (permettant uniquement de rendre compte de l'état d'une partie)</td><td colspan="3">Amélioration de l'interface graphique (interaction avec l'utilisateur)</td><td rowspan="3">Développement de l'IA en 9 semaines (tâches à définir)</td></tr>
<tr><td>Cédric Ogive</td><td>Mise en place de l'architecture (moteurs, système d'évènements)</td><td colspan="3" rowspan="2">Implémentation des règles du jeu (sauf combats)</td><td colspan="2" rowspan="2">Tests, corrections de bugs, affinement du gameplay, pour obtenir un jeu jouable humain contre humain.</td></tr>
<tr><td>Quentin Bresson</td><td colspan="2">Mise en place du système de chargement des données du jeu</td><td>Mise en place du système de combat</td></tr>
<tr><td>Sébastien Long</td><td colspan="3">Développement de l'algorithme de pathfinding, puis d'un min-max</td><td colspan="4"></td></tr>
</table>