Licence 3 Informatique
2019-2020

Mitaty Simon
Javierre Rémi

REALISATION D'UN JEU DE STRATEGIE EN JAVA

Dirigé par Mr.Goudet Olivier

université angers

Table des matières

1. Présentation du projet

1.1 Le sujet

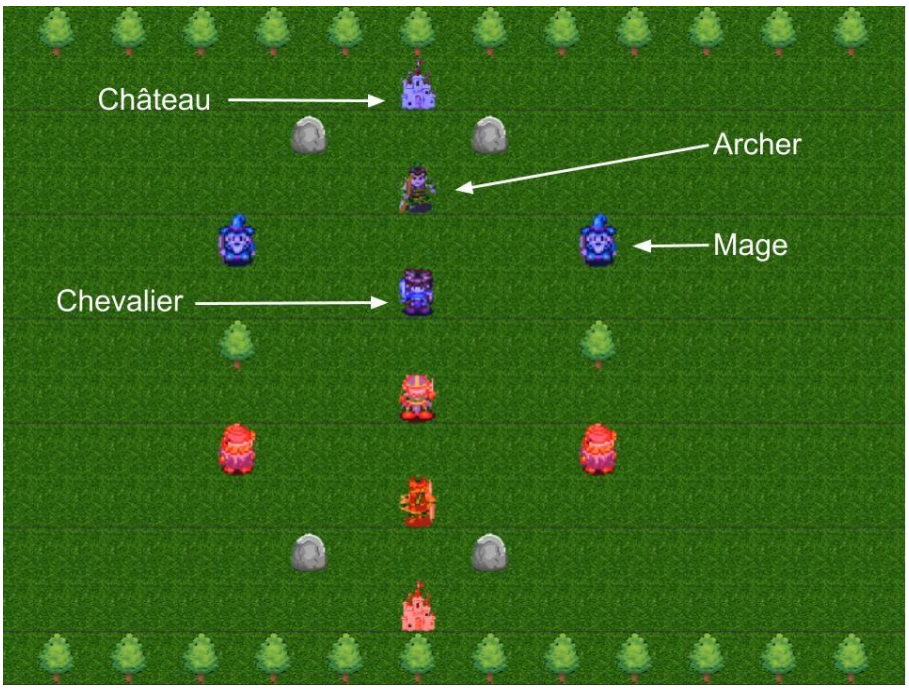
Le but de ce projet est d'implémenter un jeu de plateau tour par tour avec le langage Java. Le jeu comprend différents agents qui interagissent selon leur type (chevalier, archer, mage ou château) et selon leur équipe (bleue ou rouge). Nous avons développé l'interface graphique, le moteur du jeu et des intelligences artificielles afin de faire s'affronter des joueurs virtuels.

Pour la réalisation de ce projet, Monsieur Goudet nous a transmis un jeu (Bomberman) développé précédemment par des étudiants pour nous montrer à quoi devrait ressembler la structure de notre code et pour nous expliquer comment implémenter certaines fonctionnalités.

1.2 Le jeu

Le jeu développé est un jeu de plateau en 2D qui se déroule au tour par tour. Lors d'une partie, deux équipes se font face, l'équipe bleue et l'équipe rouge. Celles-ci sont constituées d'un château et de soldats qui s'affrontent afin de détruire le château de l'équipe adverse. Les soldats de ces armées peuvent être de différents types munis de propriétés différentes.

Sur le plateau de jeu se trouvent des rochers et des arbres, obstruant les déplacements et attaques des unités. Le château et les unités des deux équipes sont placés chacun d'un côté du plateau.



Exemple de plateau avec une équipe bleue et une équipe rouge

2. Méthodes de travail

2.1 Logiciels utilisés

Pour réaliser l'implémentation du jeu de stratégie nous avions besoin d'un environnement de développement (IDE) compatible avec Java afin de nous permettre l'accès aux librairies qu'offre le langage et ainsi réaliser un projet rigoureux. De plus, lors de notre deuxième année de formation en licence informatique, nous avions suivi une unité d'enseignement ayant pour objectif de nous initier au langage Java et la programmation orientée objet. Les professeurs chargés d'enseigner dans cette unité, nous avaient recommandé et expliqué comment utiliser le logiciel Eclipse. C'est pourquoi nous avons choisi de programmer sur cet IDE qui nous est très familier et agréable à utiliser.

Ensuite pour faciliter le travail à distance, nous avons eu besoin d'utiliser un service d'hébergement web pour permettre le dépôt et le partage de notre code. Nous utilisions déjà un site proposant ce service pour déposer nos exercices ou nos projets personnels du nom de Github. Nous avons donc par la suite créé un répertoire commun pour y déposer au fur et à mesure nos avancées sur le projet.

Pour finir, pendant la période où nous devions réaliser le projet, la situation nous a obligé à trouver un moyen nous permettant de discuter afin de nous organiser sur les tâches à réaliser. Discord est un logiciel de discussions vocales et bien plus, il possède une grande communauté et il s'avère que nous utilisions tous les deux ce logiciel pour nos propres intérêts. C'est pourquoi nous avons décidé d'utiliser Discord pour discuter et réfléchir sur la réalisation du projet.

2.2 Organisation

Le professeur responsable du projet était Mr.Goudet Olivier, nous avons donc pris contact avec lui pour qu'il puisse nous communiquer plus de précision sur le projet et savoir s'il était d'accord pour utiliser Discord pour s'entretenir. Par la suite il nous a proposé de faire des rendez-vous hebdomadaires pour discuter sur l'évolution du projet et ainsi nous communiquer semaine par semaine nos tâches à réaliser, permettant ainsi de diviser le projet en plusieurs étapes. De plus ces rendez-vous nous permettaient de présenter notre évolution au professeur mais aussi bien de lui poser nos questions sur certains détails des règles du jeu ou encore de nous aider à résoudre nos éventuels problèmes rencontrés lors de la programmation.

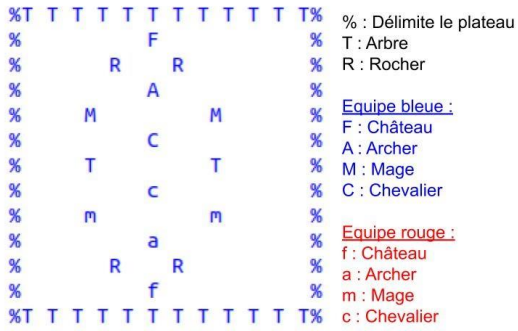
Une fois le rendez-vous fini, il était nécessaire de se partager les tâches et de se fixer sur l'organisation de chacun pour réaliser l'étape de la semaine efficacement. Ceci permettait à chacun d'évoluer à son rythme tant que toutes les tâches à réaliser étaient effectuées pour le jour du rendez-vous avec le professeur. Bien sûr nous restions tous en contact, autant entre étudiants qu'avec le

professeur, en cas d'éventuel problème afin de le résoudre le plus rapidement et éviter de rester bloqué.

3. Création et implémentation des règles du jeu

3.1 Définition des règles du jeu

3.1.1 Démarrage d'une partie



Exemple de fichier layout

Lorsque le programme est exécuté, une partie est lancée. Le lancement se fait à partir de la lecture d'un fichier layout qui représente un plateau avec les positions de départ des rochers, des arbres et des unités. La partie est une instance de la classe "Jeu" que nous avons défini et qui permet d'assurer son déroulement selon le plateau, le nombre de tours et les stratégies des équipes. Le choix de la première équipe qui joue est aléatoire.

3.1.2 Déroulement d'une partie

Les deux équipes jouent chacune leur tour. Avant chaque tour, l'algorithme s'assure que le nombre de tours maximum n'est pas atteint et qu'aucun des deux châteaux n'a été détruit. Si les conditions sont remplies, alors l'équipe qui joue choisit pour chacune de ses unités, à l'exception du château, une action que celle-ci est en mesure d'effectuer en fonction de l'environnement qui l'entoure. Les unités peuvent soit se déplacer d'une case vers le nord, le sud, l'ouest ou l'est, soit attaquer ou bien ne rien faire. Si une unité qui se fait attaquer meurt, alors elle est supprimée du jeu et disparaît du plateau.

Si une unité a plusieurs possibilités d'actions, le joueur artificiel déterminera l'action à choisir selon la stratégie qui lui a été attribuée.

3.1.3 Conditions de victoire

L'équipe qui parvient à détruire le château adverse l'emporte. Si aucun des deux châteaux n'est détruit avant le nombre maximum de tours, alors le match est nul.

3.2 Définition des agents

Un agent est représenté par un personnage qui va effectuer des actions tout au long de la partie, influant sur le déroulement de cette dernière. Dans le cadre de notre jeu, un agent est modélisé par la classe Troupes.

Chaque agent possède un attribut “PV” qui correspond à son nombre de points de vie et un attribut “degats” qui correspond aux dégâts infligés à sa cible lorsqu’il l’attaque. Il possède également un attribut “position” qui est un couple de valeurs entières (x,y) permettant de connaître la position de l’agent sur le plateau. La valeur des attributs Pv et dégâts de l’agent dépend de son type : un agent est soit un chevalier, un archer, un mage ou un château. Le type d’un agent définit ainsi sa portée d’attaque. Cependant, leurs possibilités de déplacements sont les mêmes. Les déplacements et les attaques des agents se font selon des directions, nord, sud, ouest, est mais pas en diagonale.

3.2.1 Le chevalier

Image d’un chevalier

Le chevalier est un agent disposant de 150 Pv et 40 dégâts qui se bat au corps à corps. C’est l’unité qui dispose du plus grand nombre de Pv et de dégâts mais qui est la plus vulnérable aux attaques à distance. Celui-ci ne peut en effet attaquer que les unités adverses se situant sur les cases qui l’entoure, à savoir la première case au nord de sa position, la première au sud, à l’ouest et à l’est. Il ne peut pas attaquer sur les cases diagonales.

3.2.2 L’archer

Image d’un archer

L’archer est un agent disposant de 100 Pv et 20 dégâts et qui possède une portée d’attaque de 3 cases. Ce type d’unité est fort contre les chevaliers car il a la possibilité d’attaquer sans recevoir une attaque par la suite. De plus un archer peut tirer par-dessus un rocher ou un de ses alliés mais pas par-dessus un arbre. Cependant l’archer reste une unité fragile étant donné son faible nombre de points de vie. En effet, il suffit de seulement trois attaques pour qu’un chevalier en vienne à bout.

3.2.3 Le mage

Image d'un mage

Le mage est un agent disposant de 120 Pv et 30 dégâts et qui possède une portée d'attaque de 3 cases. Il est l'unité la plus complète, en effet son nombre de points de vie lui permet de vaincre l'archer en face à face. Sa portée d'attaque ainsi que ses dégâts peuvent lui permettre de vaincre le chevalier. Tout comme l'archer, le mage peut tirer par-dessus un rocher ou un de ses alliés mais pas par-dessus un arbre.

3.2.3 Le château

Image d'un château

Le château est un agent disposant de 300 Pv mais qui ne possède évidemment pas de possibilité de déplacement ni d'attaque. Il est l'unité la plus importante car c'est sa survie qui permet ou non de remporter la partie.

3.3 Interface graphique

3.3.1 Lecture du layout

Lors de l'exécution du programme, tout commence par la création de l'interface graphique. Pour cela, comme expliqué auparavant, nous partons d'un fichier layout pour dessiner le plateau. Le programme lit ligne par ligne chaque caractère et en fonction de ce qui est lu, instancie des objets tels que des troupes ou des obstacles aux positions correspondantes. Ces objets sont ensuite ajoutés dans des listes afin de stocker leurs informations. Par exemple une liste pour les troupes de l'équipe bleue ou encore une liste représentant toutes les positions des arbres sur le plateau. Afin d'optimiser l'affichage du plateau, nous stockons les images des objets nécessaires à la création dans des tableaux. En effet un accès mémoire est 1.000.000 fois plus rapide qu'un accès disque (d'après le cours "Index et hachage" de Mme Ait El Mekki Touria). Pour finir l'affichage des objets sur le plateau se fait à l'aide des informations stockées dans les listes à leurs positions correspondantes tout en affichant sur chaque case une image d'herbe.

3.3.2 Création des équipes

Afin de différencier les équipes bleue et rouge nous utilisons des images différentes pour les chevaliers et les mages, auxquelles nous ajoutons un filtre de couleur bleu pour l'équipe bleue et un filtre de couleur rouge pour l'équipe rouge.


Mage pour l'équipe bleue


Mage pour l'équipe rouge


Chevalier pour l'équipe bleue


Chevalier pour l'équipe rouge

3.3.3 Actualisation du plateau à chaque tour

À la fin de chaque tour, il est nécessaire d'actualiser le plateau puisque chaque troupe aura effectué une action. Si l'action est un déplacement, il suffit juste d'afficher l'image de l'agent correspondant à sa nouvelle orientation (NORD, SUD, EST, OUEST) sur sa nouvelle case. Si l'action est une attaque nous devons réaliser une animation afin de distinguer l'action d'attaque de l'action STOP. Pour cela nous avions récupéré des images libres de droit représentant les différentes étapes des animations de chaque personnage.

Exemple d'animation d'attaque

3.4 Intelligence artificielle

3.4.1 Stratégies

Lors de la création du jeu on affecte à chaque équipe une stratégie. C'est pourquoi nous avons créé la classe "Strategie". Dans cette classe nous avons implémenté une méthode abstraite "coup", qui retourne une action pour une unité donnée, afin d'obliger la présence de cette méthode à chacune de ses instances. On y trouve aussi une méthode "coupspossibles" qui retourne une liste d'actions possibles en fonction d'une unité donnée et de ce qui l'entoure. Par exemple si l'unité se trouve à côté d'un obstacle ou d'une autre unité, le déplacement sur cette case est donc invalide. L'action d'attaque n'est quant à elle disponible que si le soldat est à portée d'attaque d'un ennemi. Une fois l'action de l'unité choisie, on fait appel à la méthode "jouer" qui actualise le paramètre "action" de l'unité par sa nouvelle valeur et qui modifie sa position et sa direction si la nouvelle action est un déplacement.

M.Goudet nous avait demandé d'implémenter différentes stratégies simples dans le but de les faire s'affronter et de regarder quelle stratégie s'avérait la plus efficace. Pour finir il nous avait demandé à chacun de faire une stratégie la plus intelligente possible afin de faire une petite compétition entre camarade. Ci-dessous vous trouverez toutes les stratégies implémentées et leurs fonctionnements.

3.4.1.1 Stratégie random

La stratégie aléatoire est la plus simple à implémenter. En effet il suffit, dans la méthode "coup", d'utiliser la méthode "coupspossibles" décrite au-dessus et de tirer une action aléatoire dans la liste d'actions retournée par "coupspossibles". L'action ainsi choisie sera exécutée par l'unité. Si l'action d'attaque se trouve dans la liste des actions possibles, elle est prioritaire sur toutes les autres.

3.4.1.2 Stratégie du plus proche

Cette stratégie consiste à chercher l'ennemi le plus proche de l'unité donnée et de soit s'en rapprocher, soit l'attaquer si elle est à portée. Pour trouver l'ennemi le plus proche on utilise le calcul de distance entre deux coordonnées, qui est :

$$AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}.$$

Cette stratégie possède la possibilité de se tourner dans la bonne direction pour attaquer un ennemi qui est à portée.

3.4.1.3 Stratégie focus sur le château

Le château est l'unité qui permet la victoire ou la défaite, c'est pourquoi cette stratégie cherche à rapprocher la troupe donnée du château adverse. Cependant, si lors de son trajet vers le château, la troupe rencontre un ennemi qui est à portée d'attaque, sa priorité sera de l'attaquer jusqu'à ce qu'elle ne soit plus à portée dans le cas où celle-ci fuirait.

Toutes les stratégies décrites auparavant sont des stratégies "naïves" puisqu'elles ne suivent que l'objectif qui leurs est fixé. En effet, elles ne prennent pas en compte l'environnement. Par exemple si une unité est attaquée, la stratégie ne va pas toujours lui permettre de riposter ou de se positionner dans une meilleure situation.

Dans le but d'une petite compétition nous avons implémenté une stratégie "intelligente" chacun afin de les faire s'affronter. Vous trouverez leurs détails ci-dessous.

3.4.1.4 Stratégie intelligente

La stratégie implémentée par Simon

Cette stratégie est basée sur le système de la recherche de l'ennemi le plus proche. En effet cette stratégie propose deux schémas tactiques possibles. Le premier est de chercher l'adversaire le plus proche et de chercher à l'éliminer. Le deuxième est de chercher à protéger le château s'il est en danger.

Le premier schéma cherche à tuer l'ennemi le plus proche tout en analysant à chaque tour son environnement.

Le premier élément pris en compte est le type de la troupe. Si celle-ci est un chevalier, elle aura un comportement différent que si c'est un mage ou un archer puisque leurs portées d'attaque sont différentes. En effet si la troupe est un chevalier on va chercher à avancer vers l'ennemi sans être à portée d'attaque de plus d'un ennemi puisque c'est une unité de corps à corps. Alors que pour les mages et archers on va chercher à se rapprocher de l'ennemi tout en essayant de s'aligner soit sur l'axe des X, soit sur l'axe des Y puisqu'ils peuvent attaquer à distance.

Ensuite une condition de déplacement a été imposée par le biais d'une méthode booléenne. Elle consiste à savoir si la case où la troupe veut se déplacer est dangereuse ou non. Pour cela il y a plusieurs critères, le nombre d'ennemis à portée d'attaque de la futur case et le nombre de Pv restants. Si le nombre d'ennemi est supérieur à 1, la situation est jugée dangereuse donc la case est évitée. Si le nombre d'ennemi est de 1 mais que le nombre de Pv de la troupe est inférieur ou égale au nombre de dégâts de l'ennemi (possibilité de mort), la situation est jugée dangereuse. Sinon la troupe peut avancer. Si la case permettant de se rapprocher de l'ennemi est dangereuse on regarde d'abord si les cases latérales sont sûres. Si elles ne le sont pas on peut soit ne pas bouger, soit reculer. Pour cela on prendra l'action la moins risquée tout en privilégiant de ne pas bouger plutôt que de reculer.

Pour finir, il y a une condition d'attaque qui consiste à regarder s'il vaut mieux attaquer ou fuir. Pour cela, si le nombre d'ennemi à portée est supérieur à 1 on fuit. Si le nombre d'ennemi est de 1 mais que l'ennemi est à portée de la troupe et que ses points de vies sont inférieurs aux dégâts de la troupe alors on attaque, sinon si la troupe peut mourir sans tuer l'ennemi alors on fuit. Pour les cas restants, la troupe peut attaquer.

Pour finir le deuxième schéma cherche à aller tuer l'ennemi qui attaque le château même au péril de sa vie. Dans ce cas, les conditions de déplacement et d'attaque expliquées au-dessus ne sont plus actives, le seul objectif est d'éliminer l'assaillant pour éviter la défaite.

3.4.1.5 Stratégie groupe

La stratégie implémentée par Rémi

Cette stratégie consiste à regrouper les archers vers le chevalier le plus proche afin que ceux-ci coopèrent et attaquent la même cible. Les mages quant à eux se chargent de se rapprocher de l'ennemi le plus proche de leur position dans le but de l'éliminer. L'équipe se divise donc en plusieurs groupes d'archers et de chevaliers centrés sur le comportement des chevaliers et de mages agissant seuls.

Les choix d'un mage sont faits selon son environnement. S'il peut attaquer un ennemi, alors il l'attaque. S'il ne peut attaquer mais s'il peut se faire attaquer au prochain tour, alors il se déplace sur une case où il ne pourra être la cible d'aucun ennemi, avec une priorité pour se déplacer vers son château. Cependant, s'il est déjà sur une case inatteignable pour un ennemi, alors il se dirigera sur une case sécurisée, vers le château ennemi.

Le chevalier avance vers les unités ennemies en essayant de ne jamais pouvoir se faire attaquer. Son but est de tenter d'attirer les ennemis vers lui pour ensuite faire en sorte que les archers proches du chevalier attaquent la même cible que ce dernier.

L'archer attaque à vue et se positionne sur des cases sécurisées s'il ne peut attaquer. A chaque déplacement effectué, il tente de se rapprocher du chevalier le plus proche tout en restant derrière de sorte à ne pas se rendre vulnérable. Il tentera toujours de coopérer avec le chevalier le plus proche.

Cette stratégie permet aux unités d'avancer vers le château ennemi autant que possible tout en coopérant et en minimisant les attaques subies.

3.4.2 Multithreading

Le multithreading sert à accélérer le temps d'exécution et à pouvoir lancer plusieurs threads en même temps. Dans notre cas, cela nous permet de lancer plusieurs parties simultanément et ainsi pouvoir faire des statistiques sur le nombre de victoires et de défaites lors de l'affrontement de différentes stratégies.

Equipe Rouge \ Equipe Bleue	RANDOM	PLUS PROCHE	FOCUS CHÂTEAU	GROUPE	INTELLIGENTE
RANDOM	X	Victoire bleue : 99,7% Victoire rouge : 0,02% Nul : 0,01%	Victoire bleue : 97,9% Victoire rouge : 1,9% Nul : 0,2%	Victoire bleue : 37% Victoire rouge : 61% Nul : 1%	Victoire bleue : 100% Victoire rouge : 0% Nul : 0%
PLUS PROCHE	X	X	Victoire bleue : 0% Victoire rouge : 100% Nul : 0%	Victoire bleue : 0% Victoire rouge : 100% Nul : 0%	Victoire bleue : 51,7% Victoire rouge : 48,3% Nul : 0%
FOCUS CHÂTEAU	X	X	X	Victoire bleue : 0% Victoire rouge : 100% Nul : 0%	Victoire bleue : 100% Victoire rouge : 0% Nul : 0%
GROUPE	X	X	X	X	Victoire bleue : 100% Victoire rouge : 0% Nul : 0%
INTELLIGENTE	X	X	X	X	X

Nombre de partie par affrontement : 1000 (2000 pour Intelligente vs Plus Proche car résultat très serré)
Nombre de tour maximum : 1000 (Pour éviter au maximum les égalités)

Statistique d'affrontement entre stratégie sur un plateau équitable

Classement :

1er : Stratégie Intelligente
2e : Stratégie Plus Proche
3e : Stratégie Focus Château
4e : Stratégie Random
5e : Stratégie Groupe

3.4.3 Perceptron

Suite à la création de stratégies et de comportements fortement scriptés des bots, Mr Goudet nous a présenté quelques concepts afin de créer une intelligence artificielle pour le jeu : le perceptron. Il nous a aussi fourni des classes permettant cette implémentation.

3.4.3.1 Qu'est-ce qu'un perceptron ?

Le perceptron est un algorithme d'apprentissage qui permet à une intelligence artificielle d'apprendre. Dans notre cas, cela permet à l'IA d'apprendre à jouer au jeu pour gagner un maximum de parties. Cet apprentissage se fait en jouant un très grand nombre de parties de sorte à rencontrer le plus de situations différentes possibles.
Le perceptron est représenté par un joueur (rouge ou bleu) auquel on affecte une stratégie de jeu. A chaque tour de jeu, pour chacune des actions effectuées par ses unités, le perceptron peut gagner des récompenses en effectuant des actions. Par exemple, si une de ses unités attaque une unité ennemie, il gagnera 10 points, s'il la tue 20 points. S'il attaque le château il gagnera 30 points et s'il le détruit 50 points. En revanche, s'il choisit de déplacer une unité, il ne gagnera aucun point. Le perceptron est codé de telle sorte qu'il choisisse l'action qui lui rapporte le plus de points afin de maximiser ses gains. Le but de ce système de récompense est de faire comprendre au perceptron qu'une action vaut plus qu'une autre, qu'il aura plus de chances de gagner la partie s'il choisit une action plutôt qu'une autre.

3.4.3.2 Utilisation des SparseVector

Pour que le perceptron effectue ses choix, le champ de vision de chacune de ses unités sur le plateau de jeu est limité. La taille de celui-ci est de 5 cases * 5 cases autour de l'unité, qui se trouve au centre de ces 25 cases. Cela permet au perceptron de connaître précisément l'état dans lequel se trouve ses unités : si des ennemis sont à proximité, où ils se situent par rapport à ses ennemis. Il en est de même pour les rochers et les arbres. Ce champ de vision est encodé par des SparseVector qui sont des vecteurs de taille 75. Ces vecteurs permettent de modéliser l'information sur les 25 cases qui composent le champ de vision de l'unité. Les 25 premières valeurs du vecteur décrivent les informations concernant la position des ennemis dans le champ de vision de l'unité, les 25 suivantes la position des obstacles (arbres et rochers) et les 25 dernières la position du château ennemi si celui-ci est proche. Les cases du vecteur sont soit vides soit égales à 1. En effet, pour des raisons d'économies de mémoire étant donné l'importante taille de ces vecteurs, on représente uniquement l'information des cases non vides. Ces vecteurs permettent à chacune des unités de connaître son environnement et de choisir ses actions en conséquence.

3.4.3.3 Entraînement du perceptron

Maintenant que nous avons implémenté la stratégie que doit suivre le perceptron, nous allons essayer de l'entraîner. Pour cela nous allons effectuer la simulation d'un grand nombre de partie.

Pendant chaque partie, lors de chaque action du perceptron, nous allons enregistrer des "Quadruplet". Un quadruplet est une classe qui enregistre quatre valeurs, un SparseVector correspondant à l'état avant l'action, l'action effectuée l'agent, un SparseVector correspondant à l'état après l'action et pour finir le nombre de points obtenus grâce à l'action effectuée. À la fin de chaque partie nous obtenons donc une liste de quadruplets.

Ces listes vont nous permettre d'effectuer l'apprentissage du perceptron. Pour cela nous allons utiliser une classe "LabeledSet", qui va nous permettre de stocker des exemples d'apprentissages. Un exemple contient deux valeurs, un SparseVector obtenu en fonction de l'état initial et de l'action effectuée et la récompense obtenue par l'action. Chaque exemple que nous allons passer au "LabeledSet" sera enregistré dans des listes. Une fois tous les exemples effectués nous allons utiliser la méthode "train" qui était fourni avec la classe "Perceptron". Cette méthode consiste à passer en paramètre un "LabeledSet" et à partir des exemples enregistrés le perceptron va apprendre. Ceci est appelé le Perceptron 0.

Pour améliorer le perceptron nous allons maintenant essayer de regarder au temps t+1. Pour ce faire nous allons toujours collecter des exemples d'apprentissages mais cette fois-ci pour chacun d'eux nous allons créer un nouvel exemple avec un SparseVector obtenu en fonction de l'état actuel (temps t), de l'action effectuée et une récompense qui correspond à la récompense obtenue par l'action au temps t, ajoutée à la récompense maximale possible au temps t+1. Grâce à ces exemples le perceptron va pouvoir apprendre en regardant au temps t+1 et pouvoir prévoir la meilleure action à faire. Ceci est appelé le Qlearning.

3.4.3.4 Algorithme de recherche aléatoire

Afin que le perceptron améliore son niveau de jeu, nous avons implémenté un algorithme qui permet de lancer un grand nombre de parties et de garder les meilleures parties, celle où le perceptron a acquis le plus de points. La stratégie attribuée au perceptron est la stratégie random. L'algorithme lance N parties et garde les M meilleures, avec M < N. Il complète ensuite la liste des M meilleures parties en générant N-M nouvelles parties jusqu'à obtenir un ensemble des meilleures parties de taille N. Cela permet de faire un tri des parties jouées par le perceptron de sorte à n'avoir que les meilleures pour qu'il apprenne à maximiser ses gains.

4. Planification du travail

Lors de la réalisation du projet M.Goudet Olivier nous a transmis semaine par semaine les tâches à réaliser. Nous nous les sommes ensuite réparties pour réduire la charge de travail.

Voici un planning avec le détail des répartitions des tâches. Les semaines indiquées sont du jeudi au jeudi suivant car nos rendez-vous hebdomadaires avec le professeur étaient le jeudi.

Semaine	Simon	Commun	Rémi
28 mars - 2 avril	-Création de l'IG (IG = Interface Graphique)	-Recherche des images pour l'IG -Fixation des règles du jeu	- Création de la classe "Troupes" et ses instances
3 avril - 9 avril	- Application de filtre pour la différenciation des équipes	-Implémentation des règles du jeu	-Implémentation de la stratégie random
10 avril - 16 avril	-Implémentation de la stratégie "StrategiePlusProche" -Implémentation de la stratégie "StrategieIntelligente"		-Implémentation de la stratégie "StrategieFocusChateau" -Implémentation de la stratégie "StrategieGroupe"
17 avril - 23 avril	-Implémentation du multithreading	-Renseignement sur les Perceptron et le Qlearning	-Implémentation d'une possibilité d'esquive lors d'une attaque
24 avril - 30 avril	-Gestion de l'animation des attaques -Optimisation de l'IG	-Implémentation du perceptron et du système de récompense	
1 mai - 7 mai	-Encodage des SparseVector	-Apprentissage du perceptron	-Implémentation de l'algorithme de recherche aléatoire
8 mai - 14 mai		-Apprentissage du perceptron (Perceptron 0) -Ecriture du rapport	-Implémentation des exemples et Quadruplets
15 mai - 21 mai		-Apprentissage du perceptron (Qlearning) -Ecriture du rapport	

Planning de travail

5. Conclusion

Nous avons mené à terme le projet dans les temps grâce à notre organisation et aux rendez-vous hebdomadaires. Le projet nous a permis de découvrir certaines fonctionnalités de Java, comme par exemple le multithreading ou encore la gestion de l'interface graphique. Nous avons également appris quelques notions d'intelligence artificielle et de machine learning.

6. Bibliographie

Sprites pour l'interface graphique :

- Chevalier (pour l'équipe bleue) :
  https://www.spriters-resource.com/snes/crystalbeans/sheet/2193/
- Warrior (chevalier pour l'équipe rouge) :
  https://www.spriters-resource.com/snes/crystalbeans/sheet/2199/
- Mage (pour l'équipe bleue) :
  https://www.spriters-resource.com/snes/crystalbeans/sheet/2201/
- Witch (mage pour l'équipe rouge) :
  https://www.spriters-resource.com/snes/crystalbeans/sheet/2200/
- Archer (pour les deux équipes) :
  https://www.spriters-resource.com/game_boy_advance/dynastywaradv/sheet/125870/

Machine Learning :

Cours de Mr Goudet

Perceptron : https://fr.wikipedia.org/wiki/Perceptron

Qlearning : https://fr.wikipedia.org/wiki/Q-learning