École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique

Cahier de spécification système & plan de développement			
Projet :	**Développement d'un serveur de jeu pour l'initiation à la programmation**		
Emetteur :	Nicolas MONMARCHE	Coordonnées : EPU-DI	
Date d'émission :	22 mars 2015		
Validation			
Nom	Date	Valide (O/N)	Commentaires

Historique des modifications		
Version	Date	Description de la modification

Table des matières

Cahier de spécification système

Introduction

Ce Cahier de Spécifications a pour pour but de décrire les spécifications propres au développement du projet "Développement d'un moteur/serveur de jeu pour l'apprentissage de la programmation". Le but étant que des personnes souhaitant s'entraîner au développement d'intelligences artificielles le fassent par l'intermédiaire de l'application à développer.

Ce projet a été proposé par M. Nicolas Monmarché et est également supervisé par M. Carl Esswein et M. Sébastien Aupetit.
La Maitrîse d'Œuvre est représentée par le groupe 3 PIL 2015 de Polytech'Tours, composé de :

- Ali Aalaoui
- Olivier Bureau
- Anthony Demuylder
- Nicolas Gougeon
- Jonathan Le Bonzec
- Benjamin Mauban
- Juliette Ricard
- Minghui Zhang

Contexte de la réalisation

2.1 Contexte

Un jeu en ligne est un jeu vidéo sur lequel des joueurs du monde entier peuvent s'affronter sur une même partie de jeu.
Au cours des dernières années, le jeu en ligne s'est popularisé. On compte aujourd'hui des milliers de plateformes de jeu en ligne, chacune proposant des jeux avec des gameplays très différents.
Cependant tous ces jeux ont un point commun : seuls les joueurs réels sont invités à jouer. Mais depuis quelques temps, on voit apparaître des plateformes de jeu destinées spécialement aux bots : c'est l'ordinateur qui joue.
Le challenge est très intéressant car pour pouvoir jouer au jeu, il faut d'abord réfléchir à une intelligence artificielle qui sera utilisée par l'ordinateur pour survivre dans la partie, puis programmer le bot.

Vindinium et AIChallenge sont deux plateformes de ce type. Cependant, la première présente quelques défauts de conception dans les règles du jeu et la deuxième n'est pas utilisable de manière directe. C'est pourquoi nous cherchons à créer une plateforme de jeu qui prendrait les qualités qu'on retrouve dans chacun des 2 jeux.

2.2 Objectifs

Il s'agit de créer un serveur de jeu sur lequel des bots pourront s'affronter dans des parties de jeu de type tour par tour. Le déploiement de cette plateforme permettra aux étudiants du département Informatique de s'initier à la programmation. En effet ils pourront élaborer une intelligence artificielle en programmant un bot dans le langage de leur choix. Ils pourront ensuite tester leur bot en le faisant jouer sur le serveur et disposeront d'un retour leur permettant d'améliorer leur programme.
Il faudra aussi rédiger une documentation et un exemple de bot pour que les utilisateurs puissent créer leur bot facilement.

Description générale

3.1 Environnement du projet

Le projet consiste à créer un serveur (nommé ci-après **Serveur de Jeu**) qui puisse implémenter un jeu tour par tour à un ou plusieurs joueurs. Par exemple on doit être capable de jouer une partie de jeu d'échec ou d'un équivalent à AIChallenge si le jeu est implémenté. Un "joueur" joue en réalité par l'intermédiaire d'un bot : un programme tourne sur sa machine et est capable de choisir les actions à effectuer.

3.2 Caractéristiques des utilisateurs

Dans le cas de notre exemple de jeu, l'utilisateur est une personne souhaitant jouer au jeu des fourmis. Pour cela il doit créer un bot capable de communiquer avec le moteur de jeu. Des connaissances d'algorithmiques et de programmation sont donc nécessaires pour pouvoir jouer au jeu.
De plus, le projet pourra être utilisé dans le cadre des TP d'intelligence artificielle ou de programmation. L'interface utilisateur sera donc limitée à l'application qui permettra d'inscrire un bot.

Nous développerons des exemples de bots dans différents langages afin de fournir une structure de base aux futurs utilisateurs (notamment pour faciliter l'utilisation des sockets).

3.3 Fonctionnalités et structure générale du système

Prenons l'exemple de différentes personnes souhaitant mesurer leur talent à développer une intelligence artificielle (par la suite nommée **bot**). Pour cela, on va créer un serveur (nommé ci-après **Serveur de Jeu**) qui puisse implémenter un jeu tour par tour à un ou plusieurs bots.
On doit être capable de faire s'affronter des bots sur une partie de jeu d'échec ou d'un équivalent à AI-Challenge si le jeu est implémenté. Ainsi, chaque challenger aura simplement à faire tourner un bot sur sa machine pour se mesurer aux autres.

Par ailleurs, le Serveur de Jeu doit être adaptable. Il doit être possible de le modifier facilement pour implémenter un autre jeu. Il faut donc différencier l'implémentation du jeu du Serveur de Jeu en lui-même. En revanche, un Serveur de Jeu n'implémentera qu'un seul jeu. Pour implémenter plusieurs jeux, il faudra donc cloner ce même serveur et changer le jeu.

Le Serveur de Jeu doit donc gérer la connexion de différents bots et la création de parties dans lesquelles peuvent jouer les bots. Cette insertion se fera en fonction d'un mécanisme de création de partie (que l'on appelle **matchmaking**). Il est évident que ce serveur doit pouvoir gérer différentes parties simultanément. De plus, il devra définir les protocoles de communication que le jeu utilisera pour communiquer avec les bots.

Il est important de différencier matchmaking et classement.
Le matchmaking consiste à faire jouer ensemble des joueurs ayant un classement proche. Le classement est défini en fonction de différents critères propres à chaque jeu. Le matchmaking comme le classement dépendent donc de l'implémentation du jeu.

Nous allons également nous charger de l'implémentation du moteur de jeu des fourmis. Ce jeu est grandement basé sur les règles d'AIChallenge.
Au début d'une partie, le moteur de jeu est chargé de l'initialisation. Cela consiste à définir le nombre de tours et la carte. On rappelle qu'on définit un tour comme étant l'intervalle de temps durant lequel chaque bot envoi la liste de ses actions.

Le moteur de jeu doit être capable, à chaque tour, de mettre à jour l'état de la partie (faire apparaître de la nourriture, changer les positions des fourmis, calculer l'issue des combats...).
Il recevra les actions envoyées par chacun des bots. Le rôle du moteur de jeu est de vérifier la validité de ses actions, et renvoyer les informations que chaque bot doit connaître (dans notre cas, ce que chaque fourmi du bot voit dans son champ de vision).
Ces envois et réceptions s'appuient sur les protocoles que nous aurons défini pour le serveur de jeu.

Lorsque les critères de fin de partie sont atteints, le moteur de jeu informe les bots de leur victoire ou de leur défaite. Il envoie un message de fin de partie, met à jour leurs scores pour le matchmaking, et les replace dans la file pour qu'ils puissent à nouveau jouer une partie. Le système de score sera probablement celui de l'ELO.

On mettra en place un historique de chacune des parties. Cet historique contiendra les informations permettant de retrouver l'état du jeu à n'importe quel moment de la partie.

3.4 Contraintes et outils de développement

3.4.1 Gestion de projet

SCRUM

SCRUM est une méthode agile de développement de logiciel itérative. Elle a pour but de construire le logiciel brique par brique, en commençant par les briques les plus importantes. Chaque brique apporte des fonctionnalités concrètes au logiciel pour l'utilisateur.
Nous sommes une petite équipe qui a peu d'expérience dans la gestion de projet, c'est pourquoi nous nous sommes tournés vers une méthode de travail agile.

Trello

Trello est un outil web de gestion de projets permettant d'assigner des tâches à des utilisateurs et de regrouper toutes les ressources nécessaires au développement du projet.

3.4.2 Documentation et spécifications

LaTeX

LaTeX est un outil de création de documents. Nous l'utiliserons pour nos documentations, spécifications et rapports.

Draw.io

Draw.io est un outil de dessin web permettant de créer divers diagrammes. Nous l'utiliserons pour nos diagrammes UML.

3.4.3 Développement

GitHub

GitHub est un outil web permettant l'hébergement des sources et le versioning de celles-ci via l'utilisation de Git.

Java

Java est un langage de programmation orienté objet qui présente l'avantage d'être portable. L'exécution d'un programme Java nécessite l'utilisation d'une machine virtuelle, la JVM, qui est spécifique à chaque système d'exploitation.

JSON

JSON est un format de données textuelles qui réutilise la syntaxe de JavaScript. Nous l'utiliserons pour le formatage des messages échangés entre client et serveur, fonction pour laquelle JSON est particulièrement adapté vu son faible poids.

Org.json

Org.json est une bibliothèque Java permettant de lire et d'écrire du JSON.

Maven

Maven est un outil de gestion de projet Java permettant d'automatiser la création de projets Java et de gérer l'inclusion des dépendances.

SLF4J et LOG4J

SLF4J est une interface de log à laquelle on branche la bibliothèque LOG4J pour pouvoir tracer les étapes / erreurs de notre programme.

Description des interfaces externes du logiciel

4.1 Interfaces matériel/logiciel

Le serveur de jeu aura besoin d'être exécuté sur un ordinateur relié à un réseau lui permettant de communiquer via TCP. Bien qu'être sur un réseau local uniquement fonctionne, le but final du serveur de jeu est de pouvoir être relié à Internet. Une connexion à un modem sera donc nécessaire. Le serveur d'inscription sera exécuté sur le même ordinateur que le serveur de jeu. Celui-ci et le client d'inscription devront pouvoir communiquer.

4.2 Interfaces homme/machine

Le serveur de jeu ne sera pas en interaction directe avec un utilisateur. Il n'utilisera donc pas d'interface graphique, seulement un système de log. Le serveur d'inscription ne comporte pas d'interface graphique non plus. Le client d'inscription quant à elle implémente une interface simple, demandant un nom à l'utilisateur et lui retournant un token (ou une erreur en cas de problème de réseau ou si le nom est déjà pris).

4.3 Interfaces logiciel/logiciel

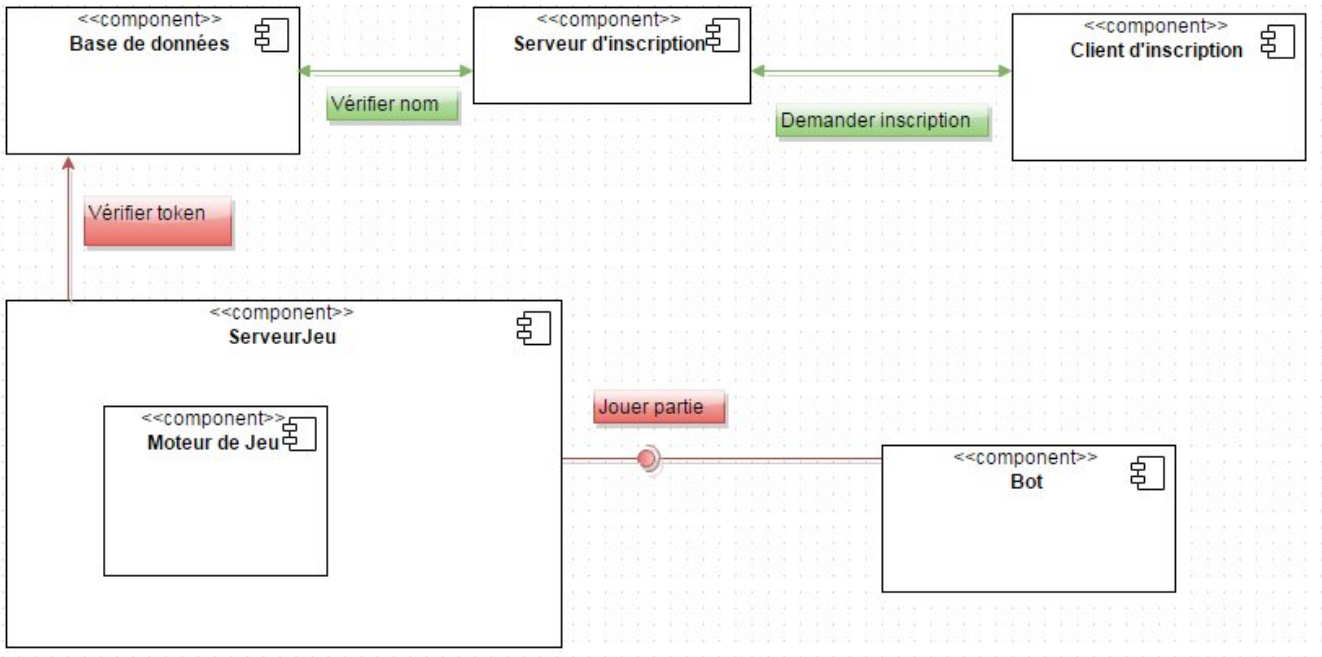
Nous utiliserons Maven pour la gestion de dépendances et SLF4J pour le système de log évoqué plus tôt. Il nous faudra également accéder de manière assez régulière à la base de donnée (au minimum à chaque inscription, connexion et fin de partie).

Architecture générale du système

Nous avons identifié 7 composants nécessaires à la réalisation de ce projet :

- un **Serveur de Jeu**, capable de gérer les connexions des bots, de créer des parties, de gérer des classements, de recevoir les actions des bots, d'envoyer l'état du jeu à chaque bot, de mettre à jour l'état d'un jeu
- un **Serveur d'Inscription**, capable de recevoir le pseudo d'un bot, de lui créer un compte dans une base de données et de lui renvoyer un jeton permettant de l'identifier de manière unique
- une **base de données**, qui stocke simplement les informations d'authentification et de score des bots
- un **client d'inscription**, qui propose une interface graphique pour communiquer avec le serveur d'inscription et qui permet à un utilisateur d'inscrire son bot
- une **implémentation** du jeu des fourmis (type AIChallenge), qui sera en fait le moteur de jeu qui sera déployé sur notre serveur de jeu et qui permettra aux bots de s'affronter dans les parties à jouer
- des **spécifications** concernant les règles du jeu des fourmis et les entrées/sorties du moteur de jeu, permettant aux lecteurs de créer leur propre bot
- un **exemple** de bot simple, qui présenterait aux utilisateurs un exemple concret pour jouer au jeu des fourmis sur le serveur

FIGURE 5.1 – Diagramme des composants

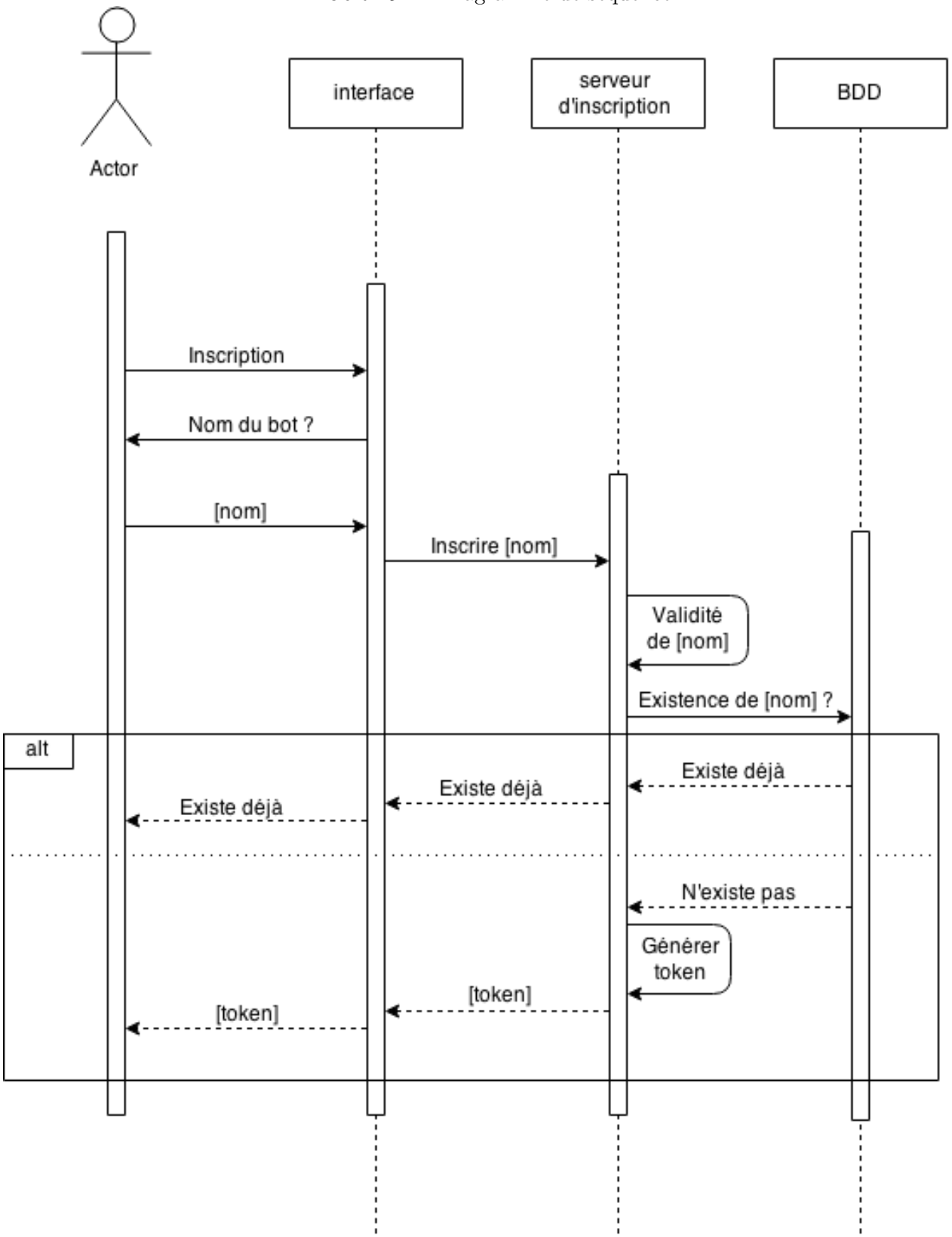


5.1 Processus d'inscription d'un bot

Les 3 composants intervenants dans l'inscription d'un bot sont le Serveur d'Inscription, le client d'inscription et la base de données.

Pour inscrire son bot sur le Serveur de Jeu, l'utilisateur doit d'abord télécharger un client d'inscription à partir duquel il peut rentrer le nom du bot à inscrire.
Le Serveur d'Inscription est chargé de tester la validité des pseudos, vérifier que le pseudo n'est pas déjà présent dans la base de données, et de générer un token unique permettant d'identifier le bot. Mais aussi, bien sûr, d'ajouter le couple nomDeBot/token dans la base de données pour que le bot puisse s'authentifier sur le Serveur de Jeu.

FIGURE 5.2 – Diagramme de séquence

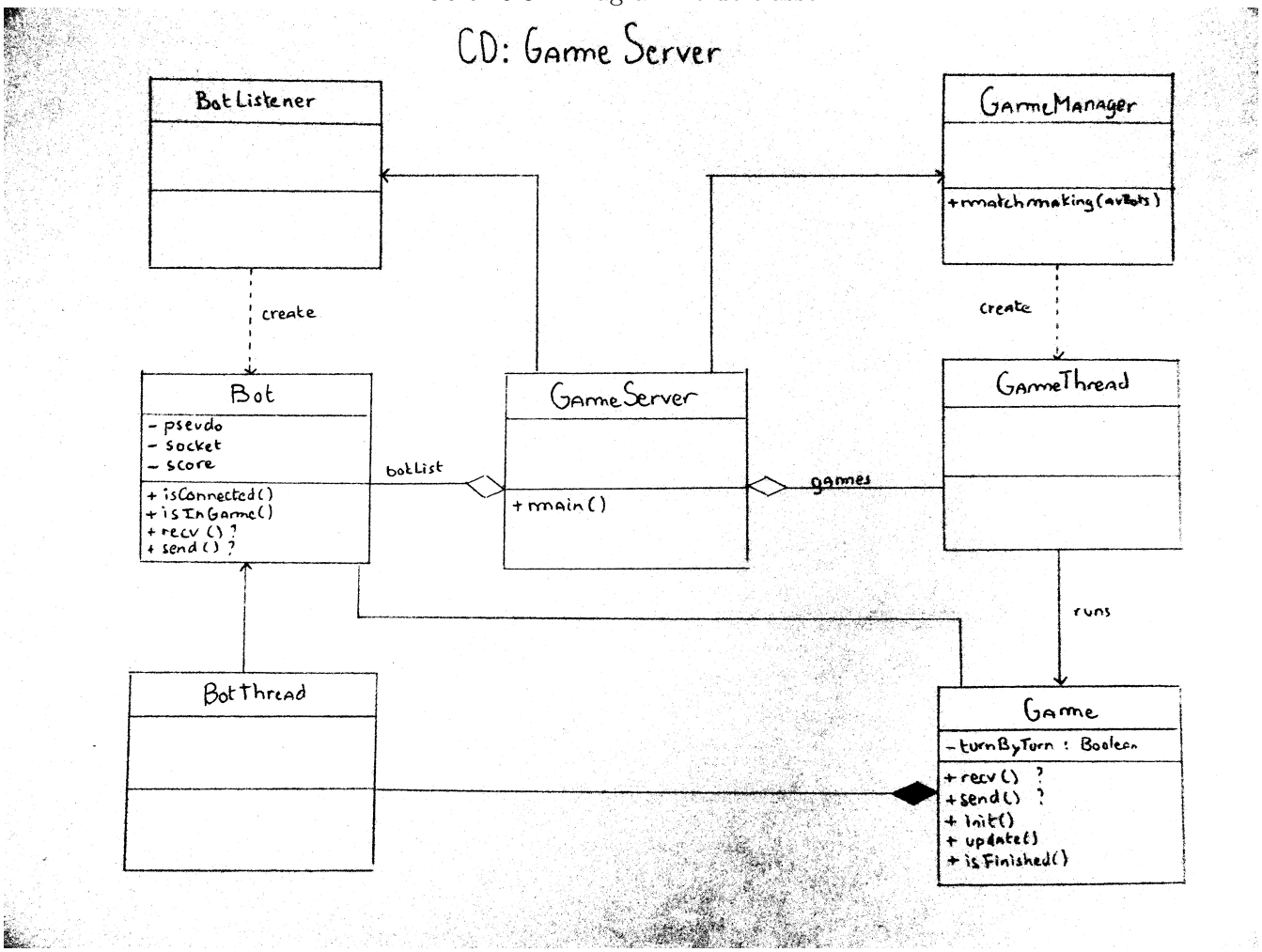


5.2 Architecture du Serveur de Jeu

Le serveur doit être capable de gérer plusieurs choses simultanément :
- l'écoute des messages d'authentification
- le système de création de partie
- les fil d'exécution de chaque partie en cours
- l'écoute des actions des bots en jeu

Il va donc falloir créer des threads pour chacun de ces éléments.

FIGURE 5.3 – Diagramme de classe



Explications sur le diagramme

GameServer est la classe principale du Serveur de Jeu.
Bot contient les informations du bot.
Le *BotListener* est chargé d'écouter les bot qui tentent de se connecter, de les authentifier et d'ajouter le Bot créé dans la liste des bots.
Le *GameManager* s'occupe de créer les parties (classe Game) à partir des joueurs disponibles. On peut surcharger cette classe pour créer un matchmaking personnalisé.
GameThread est la classe chargée de gérer une partie de jeu (voir diagramme d'activité).

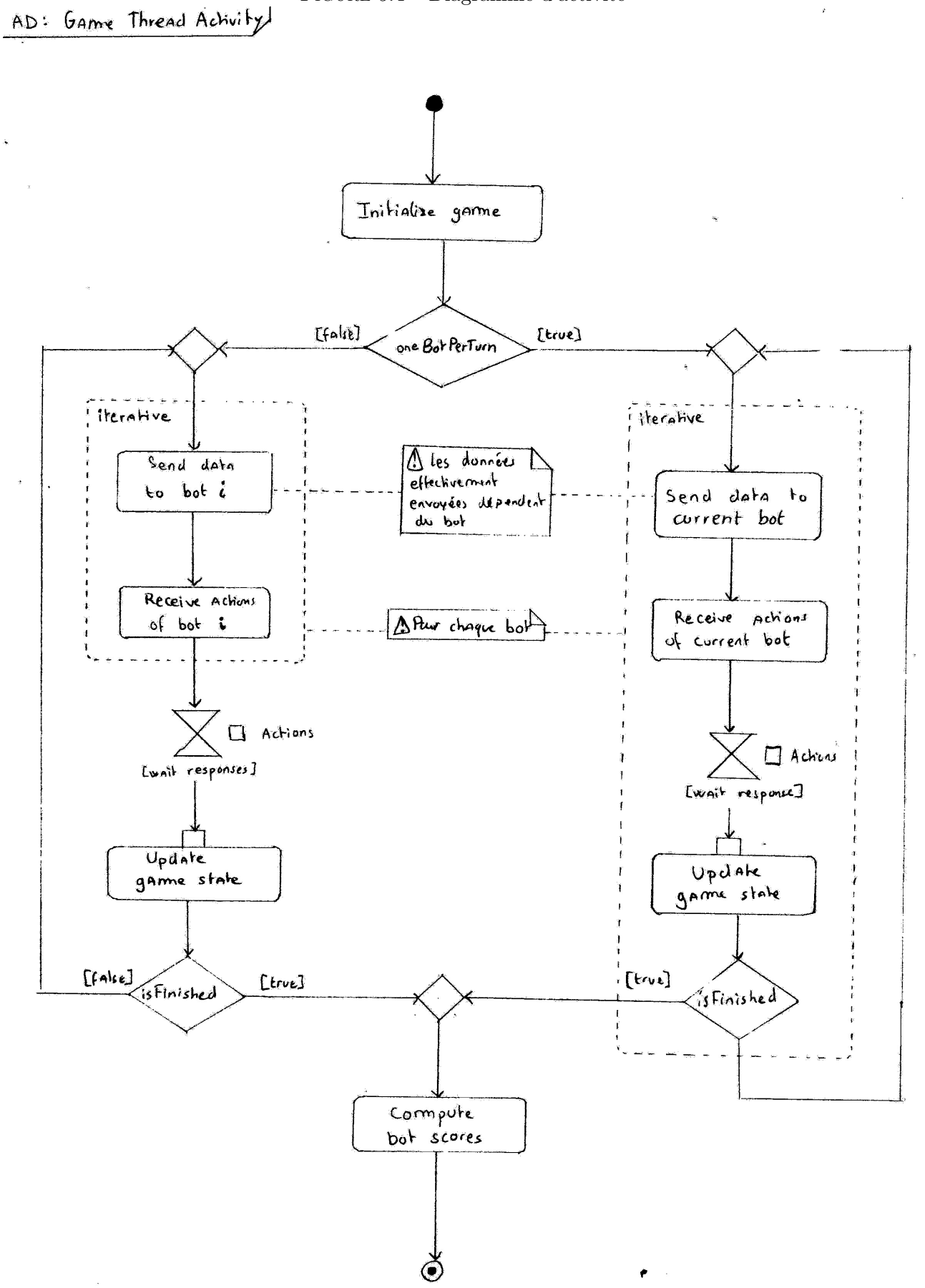
Game contient les informations d'une partie de jeu. C'est cette classe qu'il faut surcharger pour créer un autre type de jeu.
BotThread est la classe chargée d'écouter les actions des joueurs au cours d'une partie de jeu. En effet, nous avons choisi d'écouter tous les bots d'une partie simultanément afin d'accélérer les parties. Au début de chaque partie, on créé un BotThread pour chaque bot de la partie.

5.3 Déroulement d'une partie de jeu

Le déroulement d'une partie de jeu se fait différemment suivant le type de jeu considéré. Le paramètre oneBotPerTurn indique si tous les bots jouent en même temps, comme pour le jeu des fourmis que nous allons implémenter (voir branche de gauche sur le diagramme d'activité), ou si les bots jouent les uns après les autres, comme aux échecs (voir branche de droite). Dans les 2 cas le processus est similaire :

- on envoie les informations de jeu aux bots
- on attente les actions du(des) bot(s)
- on met à jour l'état du jeu
- on teste si le jeu est terminé
- si le jeu n'est pas terminé, on repart pour un tour

FIGURE 5.4 – Diagramme d'activité



Description des fonctionnalités

6.1 Inscription d'un bot

Avant de pouvoir jouer, il faut s'inscrire. Pour cela, il faut passer par un **Serveur d'Inscription** mis à disposition. Ce serveur est en charge de vérifier que le nom de bot est unique. Quand cela est le cas, il renvoie un jeton unique (généré aléatoirement) qui permet par la suite de s'identifier.
La communication entre le client (le bot qui cherche à s'inscrire) et le serveur s'effectue via une socket. Nous fournirons un exécutable qui permettra une inscription aisée d'un bot. Il suffira de passer en paramètre le nom de bot pour s'inscrire.

6.2 Gérer les connexions des bots

Cette fonctionnalité est une brique essentielle du projet : elle sera principalement en charge de gérer les flux d'informations qui transiteront entre le serveur et les bots. Elle sera capable de nous fournir diverses informations, comme par exemple la liste des bots connectés, ou le statu d'un bot (disponible, en jeu, etc).

6.3 Créer une partie : matchmaking

Ici, le système doit pouvoir "réunir" tout ce qui est nécessaire pour qu'un match puisse se dérouler. Cela se décompose en deux sous-fonctionnalités :

- Choisir une map
- Choisir des joueurs

Choisir une map : Le serveur doit pouvoir choisir une map parmi une réserve de maps ou via un générateur (optionnel). Ce choix doit pouvoir se faire selon des critères qui seront précisés plus tard (nb de joueurs par exemple, hasard, matchmaking,...).

Choisir des joueurs : Quand le serveur crée une partie, il se met en attente de joueurs. Il doit alors pouvoir gérer la connexion de joueurs.Quand un joueur se connecte il doit vérifier si celui-ci existe et créer les moyens de communication (socket). Le serveur doit pouvoir choisir des joueurs selon le matchmaking défini .
Le serveur doit enfin réunir ces joueur dans une instance de match. Le matchmaking peut être un choix au hasard parmi les joueurs connectés ou selon un algorithme permettant de gérer le classement et le temps d'attente.

6.4 Initialiser une partie

Après la création d'une partie, il y a l'étape d'initialisation. Cela consiste d'une part à définir le nombre de tours, la taille de la carte ; et d'autre part à définir l'état initial de la carte. Par exemple, les positions de chaque unité de chaque bot, celles des nids, des points d'apparition de nourriture,...

6.5 Envoyer un message au bot

L'envoi d'un message au bot, comme la réception d'un message de ce dernier se fait via des sockets. Au début de la conversation, on ouvrira une socket par bot, et on sauvegardera son identité. A chaque fois que l'on voudra envoyer/recevoir un message, il suffira d'utiliser cette socket. A la fin de la conversation, donc à la déconnexion du client, il suffira de fermer la socket.

6.6 Recevoir un message d'un bot

Pour accepter toutes les conversations, nous allons donc utiliser des threads. Les messages au cours de la partie seront principalement des informations permettant au bot de choisir ses mouvements pour les messages envoyés, et les mouvements choisis dans le cas des messages reçus.

6.7 Mettre à jour l'état du jeu

On définit l'état du jeu comme les positions de toutes les unités sur la carte à un certain moment. De même un tour de jeu est l'intervalle de temps pendant lequel chacun des bots a envoyé la liste de ses actions.
L'idée est qu'à chaque tour, l'état du jeu change. A chaque tour donc, il faut vérifier la validité des déplacements du bot. Il faut également générer les différents évènements comme l'apparition de nourriture, la destruction d'un nid, ou encore les combats. Une fois cela effectué, l'état du jeu est à jour.

6.8 Traiter les déplacements d'un bot

Peu importe le jeu, un bot se déplace. Il est donc important de savoir gérer les déplacements des bots. Cette fonctionnalité comporte aussi le traitement des déplacements impossibles.

6.9 Sélectionner des données à envoyer au bot

Le moteur de jeu doit sélectionner l'ensemble des données à envoyer à chaque joueur. Pour cela, quand nécessaire lors d'une partie, nous itérons sur les différents bots et pour chacun notre implémentation du jeu des fourmis trouve l'ensemble des obstacles, fourmis et nourritures étant dans le périmètre de vision d'au moins une des fourmis de celui-ci. Il s'agit ensuite de convertir cet ensemble au bon format (dans une chaîne de caractère au format JSON) et cette chaîne peut être envoyée au bot.

6.10 Terminer une partie et en informer les bots

Cette fonctionnalité permet de garantir la terminaison de la partie pour chaque bot, soit une fin de partie normale soit un renvoi de partie. La terminaison normale d'une partie peut dépendre de différents critères que l'on définira (un nombre de tour maximum, un nombre de points atteint, ...). Le serveur de jeu informe les bots de la fin de la partie avec un message de fin de partie ("vous avez été renvoyé pour telle raison", "vous avez gagné, voici votre score : xxx", etc).

6.11 Mettre à jour des scores

Chaque joueur possède un score. Ce score est calculé grâce à l'algorithme ELO.
ELO est utilisé dans de multiples jeu (dont le jeu d'2chec).

Le serveur utilise ce score pour chercher les joueurs ayant un niveau équivalent. Après chaque partie, le serveur va évaluer la performance de chaque bot : il va ensuite donner un score à chaque joueur. Si le joueur gagne la partie de jeu, le serveur va augmenter le score sur la base du résultat que le joueur a déjà acquis. Si le joueur perd, le serveur va diminuer le score sur la base du résultat que le joueur a déjà acquis : ce sont les scores finaux de la partie.
De plus, le montant des points est basé sur le score des autres bots. Si le bot vaincu a un score faible, le montant de points gagnés ne sera pas élevé. Le score de chaque bot est stocké dans la base de données.

Conditions de fonctionnement

7.1 Performances

On distingue deux cas lors de la connexion d'un bot :
- si il désire jouer une partie d'entrainement
- si il désire jouer une partie classée

Lorsqu'il se connecte au serveur pour jouer un bot est positionné dans une file d'attente en attendant qu'une partie soit créée.

Dans une partie d'entrainement il sera automatiquement affecté à une partie avec des bot générés par le serveur et n'aura donc pas de temps d'attente avant le lancement de sa partie.
Pour une partie classée la fonction de matchmaking attendra l'arrivée d'autres joueurs de même niveau pour créer une partie.

Au fur et à mesure que le temps passe l'intervalle de recherche des autres joueurs s'élargit. Une fois qu'un nombre de joueurs suffisant pour lancer une partie se retrouve dans le même intervalle, la partie est créée. Ainsi les joueurs seront en priorité affectés à des parties avec des joueurs ayant le même niveau.

Pour le jeu des fourmi :
Lors d'une partie le serveur enverra à chaque tour l'état du jeu à chaque joueur. Ils auront alors un temps imparti pour envoyer leurs déplacements afin de limiter la durée d'une partie.

7.2 Capacités

Il faudra déterminer un nombre maximum de joueurs qui pourront se connecter.

7.3 Modes de fonctionnement

Le serveur devra pouvoir tourner sur les machines de l'école.

7.4 Sécurité

Lors de son inscription le joueur reçoit un token. Comme aucune donnée confidentielle ne sera stockée on ne demandera pas de mot de passe. Seul le token sera suffisant pour se connecter.

7.5 Intégrité

Pour gérer la déconnexion d'un joueur lors de sa recherche de partie on interrogera les joueurs avant le lancement de la partie.
Si la déconnexion survient lors de la partie on gérera cette déconnexion comme si le joueur était exclu de la partie pour n'avoir par répondu durant le temps imparti.

Plan de développement

Découpage du projet en User Stories (tâches)

1.1 L'utilisateur est capable de créer un bot qui sait jouer au jeu des fourmis

Cette user story implique la création d'une documentation qui explique :

- comment jouer au jeu des fourmis (principe, règles de jeu...)
- comment communiquer avec le serveur
- comment s'inscrire, se connecter sur le serveur

On estime que cette tâche est longue et importante car sans documentation, les utilisateurs seraient obligés de se plonger dans le code pour créer leur bot et par conséquent, il n'y aurait que très peu de bot en ligne. Une documentation claire et concise a l'avantage de motiver les lecteurs pour créer leur bot.

Cette tâche ne dépend d'aucune autre tâche. Cependant, il est préférable de rédiger la documentation à la fin au cas où il y aurait des changements pendant le développement de l'application.

1.2 Un bot peut s'inscrire

L'utilisateur doit être câble d'inscrire un bot. Pour cela il utilisera un exécutable créé par nos soins. Cet exécutable ouvrira une fenêtre permettant de rentrer le nom du bot désiré. Il ira ensuite interroger le serveur d'inscription pour vérifier que le nom du bot n'est pas déjà pris. S'il est déjà pris, l'utilisateur sera invité à choisir un autre nom de bot. Dans le cas où ce nom est disponible, le bot sera enregistré dans la base de données et un identifiant unique sera retourné à l'utilisateur. Cet identifiant (token) permettra d'identifier le bot de manière unique lorsqu'il se connectera au Serveur de Jeu.

1.3 Un bot peut se connecter et communiquer avec le Serveur de Jeu

Afin d'assurer un moyen de communication avec le bot, le Serveur de Jeu utilisera le protocole TCP, et tous les messages transférés dans un sens comme dans l'autre devront être au format JSON. Pour ce faire, il utilisera un serveur TCP permettant d'écouter les connexions entrantes sur un certain port.
À chaque fois qu'une connexion est faite, le Serveur de Jeu attend un message sur cette connexion, censé contenir le token du bot envoyé par celui-ci. Ceci permet de confirmer l'identité du bot. Ce premier message contient également le type de jeu désiré par le bot ("training" ou "classique"). Ensuite, afin d'assurer des communications asynchrones avec les différents bots connectés, le Serveur de Jeu utilise au moins un thread par bot servant exclusivement à écouter ses messages quand nécessaire.

1.4 Un bot peut rejoindre une partie (matchmaking)

Une fois qu'il est connecté, l'utilisateur doit pouvoir rejoindre une partie. Pour cela nous allons devoir mettre en place un système de création de partie (parfois désigné comme matchmaking). Ce système consiste à faire en sorte que les bots ayant un niveau similaire se rencontrent. Il s'occupera également de choisir la carte sur laquelle se déroulera le match.

Faire s'affronter des bots ayant un niveau proche présente une contrainte : le temps d'attente. En effet si on attends d'avoir des bots ayant un classement quasiment identique, cela peut s'éterniser. Pour éviter cela, nous allons avoir recours à un algorithme qui élargit peu à peu le champ de recherche. A l'arrivée d'un bot on essaiera de le faire s'affronter contre son "voisin" de classement. Puis, au fur et à mesure que le temps s'écoule, on élargit ce voisinage. On pourra fixer un temps limite au bout duquel il peut affronter un bot de n'importe quel niveau. Ainsi, il sera possible de limiter le temps d'attente total d'un bot à (au maximum) la somme des temps au bout desquels les bornes extrême de chaque bot est atteinte.

1.5 Un bot peut jouer en mode training

On se propose de faire un mode d'entraînement. Ce mode "bac à sable" permettra aux utilisateurs de tester le bon fonctionnement de leur bot contre des Intelligences Artificielles "basiques". Le but sera donc de créer un match confrontant le bot de l'utilisateur et les bots d'entraînement disponibles sur le serveur (et aucun autre bot).

1.6 Un bot peut déplacer ses fourmis

Cette tâche à pour but de mettre en place le déplacement des fourmis. Cela implique :
- définition d'une syntaxe JSON pour les actions du bot
- vérification des actions envoyées (la fourmi va-t-elle dans un mur ?)
- traitement des positions des fourmis côté serveur
- envoi au bot des nouvelles positions de ses fourmis ainsi que des informations de la carte visibles par celles-ci
- traitement des positions des fourmis côté serveur
- envoi au bot des nouvelles positions de ses fourmis ainsi que des informations de la carte visibles par celles-ci
- test du déplacement des fourmis

Pour réaliser cette user story, nous allons devoir implémenter la capacité des fourmi a pouvoir ce déplacer selon les règles du jeu qui auront été établies. Pour cela nous allons devoir mettre en place un protocole permettant au serveur de jeu d'effectuer ces déplacements.
Par exemple nous créerons une instruction afin de demander le déplacement d'une certaine fourmi vers la droite. Ce protocole sera conçu et interprété en JSON. La première partie de cette tâche sera donc un travail de groupe afin de concevoir le protocole de déplacement. La deuxième partie permettra ensuite d'implémenter ce protocole afin que le serveur de jeu sache l'interpréter. Enfin, une phase de tests unitaire sera nécessaire afin de s'assurer du bon fonctionnement de chaque instruction dans le maximum de cas(déplacement, d'une fourmi dans un mur, dans une autre fourmi, dans un nid etc).

1.7 Un bot peut récupérer de la nourriture

Un des objectifs principaux pour la victoire est d'augmenter son nombre de fourmis. Pour cela le bot a la possibilité d'envoyer ses fourmis récolter de la nourriture.

La nourriture est disposée symétriquement sur la carte afin de préserver l'équité entre les joueurs. Elle apparait sur des points de spawn fixes durant toute la partie.

Optionnellement, les fourmis devront ramener chaque unité de nourriture jusqu'à leur propre nid afin qu'il soit comptabilisé. La fréquence d'apparition de nourriture sera gardé secrète des joueurs mais sera la même pour tous les points de spawn d'une map.

1.8 Un bot peut détruire un nid adverse

Afin d'endiguer la naissance de nouvelles unités ennemies, il peut être intéressant de détruire ses nids. Pour cela, le bot doit simplement faire en sorte qu'une de ses unités se trouve "sur" le nid ennemi. Autrement dit, amener une unité alliée à l'exacte position d'un nid adverse permet de supprimer ce dernier.

1.9 Un bot peut combattre

Le jeu permet l'affrontement entre les unités. le système mis en place est simple, chaque fourmi possède une zone de combat tout autour d'elle. Lorsqu'une fourmi adverse se retrouve dans le champ de combat d'une autre les deux fourmis sont détruites. Dans le cas ou il y a plus de fourmis du camp adverse dans le champ de combat de la fourmi celle-ci s'auto-détruit.

1.10 Un bot peut être renvoyé s'il ne respecte pas les règles du jeu

Dans le cas d'un renvoi de partie, les fourmis du bot sont conservées sur le terrain et ses points sont tout de même calculés en fin de partie, mais le bot ne reçoit plus l'état de la carte à chaque tours et ne peut plus envoyer ses mouvements. Le bot peut être renvoyé dans plusieurs cas : sa connexion avec le Serveur de Jeu a été fermée, le bot a mis trop de temps à envoyer sa réponse ou la réponse n'est pas correcte (pas au format JSON, ne contenant pas les bonnes informations, ...).

1.11 Un bot peut gagner une partie

Un bot peut gagner une partie de plusieurs manières. En effet, un timer est mis en place afin qu'une partie ne dure pas plus longtemps qu'un certain temps défini. Au bout de ce temps, le bot ayant le plus de points est désigné comme le gagnant de la partie. Si avant la fin du temps imparti un bot réussi à « éliminer » tous les autres bots alors il gagne la partie.

1.12 Un bot peut récupérer l'historique d'une partie

Ici, le travail à réaliser est l'envoi de l'historique d'une partie aux bots y ayant participé. Pour cela il faut d'abord sauvegarder le déroulement de la partie au cours de celle-ci. Il a été décidé que nous stockerons les différents changements d'états de la partie et non les différents états de la partie. Un changement d'état représente n'importe quel élément mobile du jeu des fourmis : fourmi, nourriture, nid. Seul l'état initial sera sauvegardé. Le moyen de persistance que nous avons choisi est la sauvegarde sous forme de fichier.

Si l'on est dans le cas ou chaque bot joue l'un après l'autre, c'est après avoir exécuté toutes les instructions du bot que les changements seront sauvegardés. Dans le cas ou tous les bots jouent en même temps, c'est après avoir effectué tous les déplacements du tour que la sauvegarde sera effectuée.
De plus, il va falloir utiliser une méthode permettant de transcrire efficacement les changements d'états qui pourront survenir. Chaque partie sera donc résumée dans un fichier via un état initial et une liste de

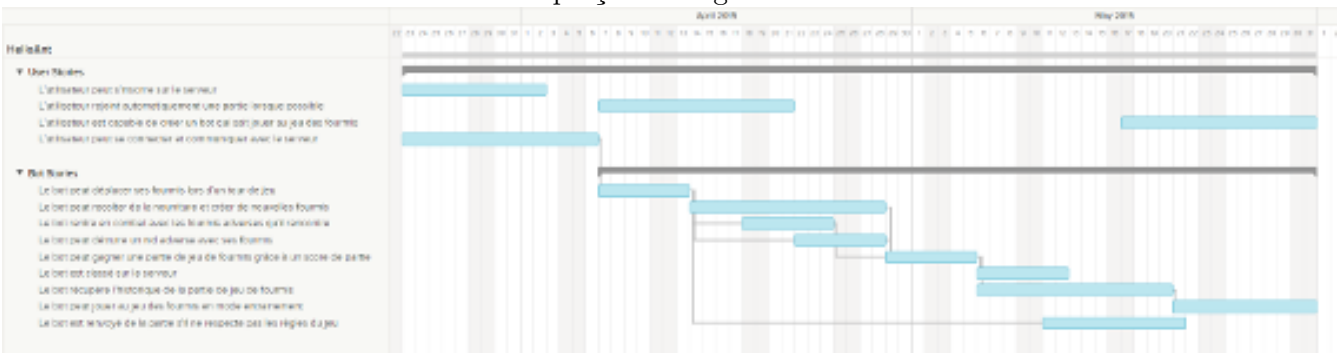
changements d'états.Il faudra donc concevoir les fonctions nécessaires à cette sauvegarde et procéder au tests unitaires sur celles-ci.

Dans un second temps, nous devrons envoyer cette liste à tous les bots ayant participé à la partie. Comme le reste des communications, l'envoi de ce fichier se fera par socket. Les sockets utilisés seront ceux établis avec les bots durant la partie. De même, une phase de tests sera nécessaire pour valider cette user story.

Planning

Vous pourrez trouver notre diagramme de Gantt **ici**.

FIGURE 2.1 – Aperçu du diagramme de Gantt



Glossaire

Ci-dessous se trouvent les définitions des différents termes utilisés. Les mots sont sous la forme suivante : motEnAnglais (motEnFrançais) : définitionEnFrançais.

Client (Client) Machine physique sur laquelle est lancé le bot
Bot (Bot) Programme sur la machine client qui joue à différentes parties
Token (Jeton) Identifiant unique associé à un seul bot. Chaque bot possède son propre jeton
Game Server (Serveur de Jeu) Programme réseau permettant l'interface entre les bots et le jeu
Game Engine (Moteur de Jeu) Implémentation des règles, permet de changer l'état de la partie
Inscription Server (Serveur d'inscription) Programme réseau qui permet au client de s'inscrire. On créera une application client exprès afin que son inscription soit aisée
Game (Jeu) Similaire à ce qu'on sous-entend dans la vie réelle.
Exemple : un jeu d'échec
Match (Partie) Intervalle durant lequel différents bots s'affrontent.
Exemple : une partie de jeu d'échec
Action (Action) Décision prise par le bot concernant les déplacements de ses unités.
Exemple : Pour le jeu des fourmis, c'est l'ensemble des mouvements des unités
Match State (Etat de Jeu) A un certain moment, les positions de toutes les unités sur la carte
Match Turn (Tour de Jeu) Intervalle de temps pendant lequel chacun des bots a envoyé la liste de ses actions. Cela peut différer selon le type de jeu. Exemple : Sur un jeu d'Echec, un tour correspond à un déplacement du joueur A et au déplacement du joueur B (retour au joueur A)
Player (Lecteur) Comme un lecteur musical, est capable de lire un historique d'un match afin d'avoir un rendu visuel.
Replay (Historique) Sauvegarde de tous les mouvements / toutes les positions des unités de chacun des bots au cours de la partie
Mouvement invalide Action du joueur considérée par le joueur comme interdite par les règles.
Exemple : aux échecs, déplacer un pion vers l'arrière est un mouvement invalide
Map (carte) Ensemble de cases servant d'environnement aux bots. Ex : dans le cas du jeu des fourmis, la map est un tableau en 2 dimensions
Matchmaking (création de partie) processus permettant d'attribuer des bots et une map à un match
Score (Score) Classement du bot pour un jeu. Peut être calculé sur le principe de l'ELO
Match Score (Score de partie) Nombre de points de points qu'a accumulé un bot lors d'une partie d'un jeu