UNIVERSITÉ
DE LORRAINE

Collaboration entre agents artificiels

Julien BERNARD

—

Thomas ROBINEAU

—

Hugolin ROUYER

—

Nicolas RUSSO

Tuteur : Vincent THOMAS

Année universitaire : 2023/2024

Lien: github.com/ThomasRbn/PTUT_BERNARD_ROBINEAU_ROUYER_RUSSO

Table des matières

I. Présentation du projet

Ce document final traitera de manière linéaire toute la réalisation ce projet tutoré, de la conception, jusqu'à la livraison du produit final.

L'objectif principal de ce projet est de concevoir un environnement de simulation inspiré du jeu "Overcooked", dans lequel plusieurs agents artificiels ou non, doivent collaborer pour résoudre une tâche commune, à savoir préparer et servir des plats à des clients. Ce projet est axé sur la recherche de solutions permettant à ces agents de travailler ensemble de manière efficace et coordonnée.

1. Création d'un Simulateur "Overcooked"

Overcooked est un jeu de coopération et d'adresse dont l'objectif est la réalisation de plats et l'envoi aux clients. La réalisation de plat correspond au traitement des aliments et de leur association. Au fur et à mesure du jeu, nous avons accès à des niveaux de plus en plus complexes.
L'intérêt du jeu réside entre autres dans le besoin permanent d'interaction et de coordination entre les cuisiniers lors de la préparation de plats dans un environnement difficile parsemé d'embûches. Se servir de Overcooked comme source d'inspiration pour le développement de ce projet, est que ce jeu est multijoueur et nécessite une bonne gestion des tâches pour définir qui fait quelle tâche.

Le simulateur de jeu reflète donc le concept de ce jeu. L'environnement simulé comprend, pour le moment, des éléments tels que des ingrédients, des plats à préparer, une carte avec les chemins pour accéder à chaque élément, des planches à découper, des plaques de cuisson pour cuire et un dépôt.

L'application est réalisée en java et comporte une interface graphique JavaFX. Avec la possibilité de jouer à plusieurs joueurs. L'application intègre des agents artificiels. L'objectif du jeu est que plusieurs agents agissent ensemble pour résoudre une tâche commune (ici préparer et servir des plats à des clients).

2. Fonctionnement du jeu

Les joueurs jouent leur tour respectif en même temps dans l'optique de réaliser un ou plusieurs objectifs (les plats). Il est nécessaire que les joueurs présents effectuent une action qui peut être :

- HAUT,
- BAS,
- GAUCHE,
- DROITE,
- PRENDRE,
- POSER,
- UTILISER.

Ils peuvent suivre la progression du jeu grâce à une interface graphique contenant les informations nécessaires et une carte sous forme de grille représentant les différentes cases du terrain.

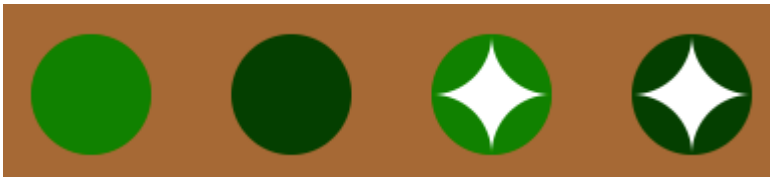
3. Une interface graphique

L'interface graphique créée en JavaFX se compose de plusieurs éléments ordonnés de la façon suivante.

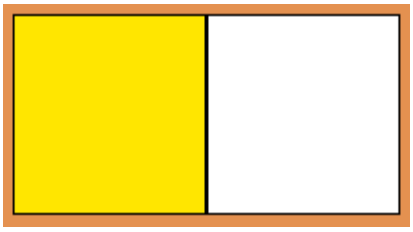
Un menu permettant la configuration de la partie, avec un sélecteur de niveau parmi une liste de niveaux préconstruits, un sélecteur de joueurs qui permet de choisir entre différents types d'Agent Artificiel (AA[1] V1[2], AA V2, Humain autonome) et un joueur humain. A noter que le nombre de joueurs maximum par niveau est fixé à 2 et n'est modifiable que dans le fichier du niveau. Également, il est possible de choisir la durée de la partie en entrant le nombre de secondes dans le champ approprié.

[1] AA: Agent Artificiel.
[2] V1: Les deux versions des agents artificiels seront définis par la suite.

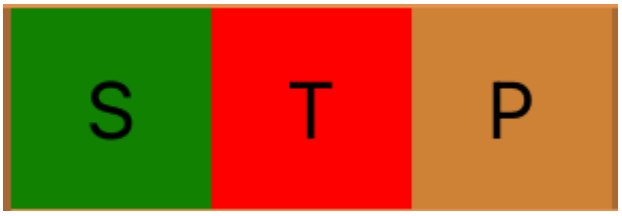
Pour l'affichage de la partie, l'écran est divisé en 3 zones distinctes.
On retrouve d'abord la grille représentant le niveau avec des carrés/cercles de couleurs différentes symbolisant les éléments du jeu.



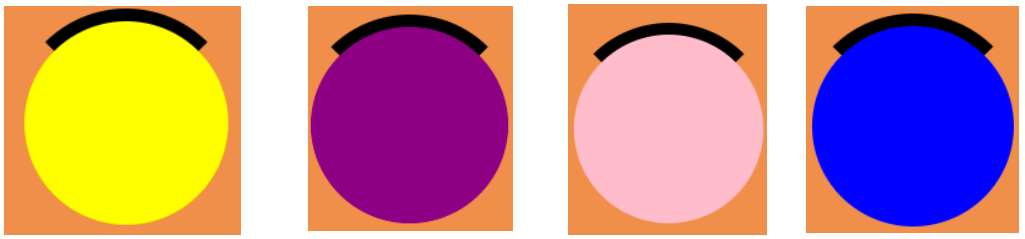
Aliments : Salade crue, Salade cuite, Salade coupée, Salade coupée et cuite



Transformateurs : Plaque de cuisson, Planche à découper



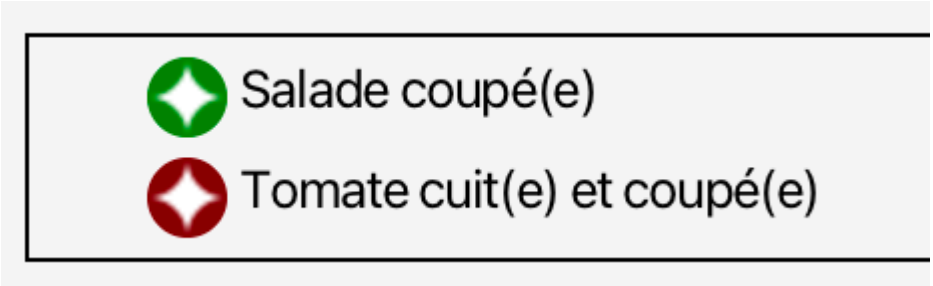
Générateurs : Salade, Tomate, Pain



Joueurs : Joueur Automate, Joueur Humain, Joueur AA v2, Joueur AA v1

Le cône noir des joueurs correspond à la direction où ils regardent.
S'ils portent quelque chose, le cercle de l'aliment correspondant se superpose au cercle du joueur.

On retrouve également un bandeau donnant les différentes informations sur la partie en cours comme le temps restant et le nombre de points accumulés jusque-là. Également, on retrouve des indicateurs de décision représentés par un point vert si le joueur correspondant a choisi l'action qu'il souhaite faire, et rouge sinon.



Un affichage sur la partie gauche de l'application indique les différentes recettes acceptées par le dépôt sur le niveau sélectionné, avec les aliments qui la composent.

L'ensemble de ces informations sont présentes dans un fichier texte. L'application comporte un système de lecture du fichier qui initialise les données en début de partie, que ce soit la carte ou les recettes.

4. Déroulement de la partie

Le jeu débutera avec l'initialisation du niveau, à partir d'un fichier texte, qui sera suivi de la demande d'action aux joueurs. Les calculs liés à l'agent artificiel se font à chaque début de tour puisqu'il doit réaliser une action à ce moment. Les actions possibles sont donc : un déplacement, une action (ex: prendre ou poser un objet, découper,...) ou ne rien faire. Une fois que toutes les actions des joueurs sont reçues, l'application réalise les actions de tous les joueurs ainsi que leur traitement associé. Le niveau se termine lorsque le temps, défini avant le début du jeu, est écoulé.

5. Multijoueur à 2 Joueurs

L’application est capable de gérer 2 joueurs indépendamment l’un de l’autre. Au lancement de la partie, après avoir sélectionné le niveau, le programme demande à la personne de choisir le type de joueurs qui seront ajoutés au jeu.

Une fois que la partie commence, le décompte de temps est lancé et tous les joueurs doivent sélectionner l’action qu'ils souhaitent faire au prochain tour. Les actions ne sont pas résolues avant que tout le monde n’ait fait son choix. Pour rappel, un cercle de couleur permet de savoir qui a choisi son action.

6. Intégration d’un Agent intelligent

Comme dit précédemment, il est possible d’intégrer un agent artificiel parmi les joueurs. Ce dernier calcule son action, en partant de l’état actuel, à chaque fois qu’il lui est demandé. Pour ce faire, il génère les données qui lui sont nécessaires puis, à l’aide des algorithmes A* et UCS, calcule la prochaine action qu’il lui faudra faire et la retourne. Nous avons développé deux Agents artificiels avec un fonctionnement inverse qui sera développé plus tard.

II. Analyse

Initialement, il était prévu d'avoir une interface graphique avec un menu et des agents artificiels centralisés et décentralisés. Au cours des itérations, nous nous sommes rendu compte qu'il serait compliqué de réaliser les deux versions. Nous avons donc, après discussion avec notre Tuteur, décidé de ne faire que la version décentralisée. Cette version était l'objectif final et elle correspondait plus à notre architecture du moment et ne nécessitait pas de changement.

À l'heure actuelle, à la fin de l'itération 4, nous disposons d'une interface graphique avec menus permettant de sélectionner le niveau, les joueurs et le temps de jeu. Une fois le jeu démarré, nous pouvons voir quel joueur a choisi son action, combien de temps il reste avant la fin de la partie ainsi que les plats à réaliser pour le niveau en cours.
Concernant les Agents artificiels, nous avons deux versions avec chacune leurs spécificités. Ces derniers seront développés par la suite.

1. Evolution de l'application

Itération 1

Au cours de la première itération, nous voulions pouvoir préparer l'application pour débuter un développement plus complexe de l'application par la suite. Il nous fallait donc mettre en place une structure extensible et nous permettant d'observer ce que nous réalisions.

Pour ce faire, Thomas et Hugolin ont commencé par réaliser une interface graphique primaire comportant les fonctionnalités suivantes :

- Sélection d'un niveau par l'utilisateur parmi une liste, au sein du code.
- L'utilisateur doit pouvoir distinguer les différents éléments graphiques facilement (une couleur/icône pour une surface).
- L'orientation du joueur doit être prise en compte et jouable (Si la case sur laquelle le joueur veut se déplacer est une surface, alors il ne bouge pas et s'oriente face à elle).
- Lorsque l'utilisateur prend un objet d'un générateur, il doit le tenir dans les mains (icône de l'objet superposée à celle du joueur).
- Quand un joueur essaye de déposer un objet sur une surface invalide (un distributeur ou une surface déjà occupée), rien ne doit se passer.

Pour permettre certaines de ces fonctionnalités, il nous a fallu réaliser une structure de données pouvant servir à l'interface graphique, mais également aux simulations des agents artificiels :

- Mise en place d'un système permettant de transformer un fichier texte en jeu de données, chaque élément étant représenté par un caractère. Cela permet une création et une modification beaucoup plus simplifiée des niveaux.
- Une classe contenant l'ensemble des données de notre application, à savoir les objets fixes, les objets déplaçables, le/les plats buts et une liste de joueurs. Mais pour pouvoir éviter les effets de bord, des constructeurs par copie ont dû être mis en place. Cela a permis d'éviter des problèmes, notamment au niveau des simulations des agents artificiels.
- Un système de recettes et de plats. Un plat est un ensemble d'aliments, et un Aliment a un nom et un état (cru, cuit, coupé, cuit-coupé). Un état hérite de la classe Aliment et à un Aliment comme attribut. Cela permet d'avoir plusieurs états qui s'ajoutent à un seul aliment, pour avoir au final un Pain - Coupé - Cuit.
- Création des joueurs et de leurs structures de données. Chaque type de joueur hérite de la classe joueur pour la mise en place du patron stratégie permettant de faciliter leur gestion. Il est donc possible de demander une action à un joueur, quel que soit son type.

De son côté, Julien, en plus d'aider à la réalisation de la structure de données, réalise une version 0 de l'agent artificiel fonctionnant avec des données fictives. Cette version fonctionne comme une version miniature de ce qui était envisagé par la suite : lors de l'appel de demanderAction, nous actualisons les données, ici totalement fictivement, mais par la suite, à partir des données réelles.
Parmi ces données, nous avons un but, qui est de déposer un plat en particulier. Dans son fonctionnement, l'AA (agent artificiel) simule des déplacements pour aller chercher un aliment (dans cette version simpliste) et le transporter jusqu'au dépôt. Une fois cette version faite, il a été possible de commencer à fonctionner avec les vraies données.

Nous pouvions donc commencer à faire des simulation en générant des données qui seront proches des données finales. Pour réaliser les simulations, nous utilisons l'algorithme A* qui parcourt toutes les actions possibles à ce moment (Haut, bas, gauche, droite, prendre).

L'un des points essentiels de A*, c'est son heuristique. Au cours de l'Itération 1, seule une version simpliste de calcul de l'heuristique a été mise en place, mais par la suite, il nous a fallu l'optimiser. L'heuristique sert à orienter la recherche de chemin dans la bonne direction. À ce moment, le but était de prendre une tomate et de la

déposer. L'heuristique réduisait lorsqu'on se rapprochait de la tomate (sans en avoir), puis réduisait lorsque le joueur avait pris la tomate et se rapprochait du dépôt.

Itération 2

Pour commencer l'itération 2, Julien a réalisé un nettoyage des fichiers. Lors de la première itération, nous n'avions pas fait attention à bien répartir les fonctionnalités dans différents fichiers, ce qui a produit plus de 500 lignes dans DonneesJeu. Au début de l'itération, il a donc fallu répartir les différentes fonctions, telles que la comparaison de deux DonneesJeu, copie de DonneesJeu (dupliquer DonneesJeu sans produire d'effet de bord), pour les simulations de l'agent artificiel. Ainsi que d'autres fonctions telles que la création des données et ce qui concerne les actions des joueurs.

Nicolas a effectué des rangements au sein de l'interface graphique, notamment en créant des vues pour chaque élément affiché à l'écran. Cela inclut donc les blocs, le sol, les aliments et les joueurs. Cela permet de réutiliser ces éléments, au lieu de tout devoir configurer plusieurs fois en dur. Cela était le cas lors de la première itération, car nous avions mis en place une interface très minimaliste où des classes externes n'étaient pas nécessaires.

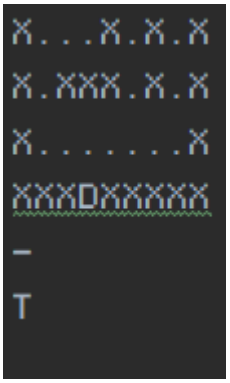
Pour fonctionner, l'agent artificiel a eu besoin de récupérer certaines données ou pour pouvoir en modifier. Hugolin, Nicolas et Julien ont travaillé sur cette section pour adapter légèrement les données aux nécessités des algorithmes de l'agent artificiel. En effet, pendant que Julien développait ce dernier, il a eu besoin de certaines données ou de changer leur fonctionnement. Parmi celles-ci, on peut noter les changements nécessaires dans "isLegal", la méthode permettant de savoir si une action est légale dans un état pour un joueur. Pour garantir le bon fonctionnement de ces méthodes, un ensemble de tests robustes ont été mis en place. Cela permet de garantir le bon fonctionnement du corps métier, de déboguer plus facilement les agents artificiels et d'anticiper d'éventuels conflits avec les nouvelles fonctionnalités.

Un des problèmes rencontrés a été le système de plat, qui avait un ensemble de données très lourd et difficilement clonable pour les simulations de l'agent artificiel. De ce fait, Nicolas a de son côté simplifié le système de recettes. En effet, les états ne sont plus des classes qui suivent une approche de décorateur, mais bien des entiers. Cela allège fortement le processus de création et de clonage, et cela reste cohérent avec notre structure.

Il y a eu également l'ajout des transformateurs au sein du jeu, à savoir les planches à découper et les plaques de cuisson. Les transformateurs vont modifier le plat instantanément. Le principe des transformateurs est de pouvoir modifier l'état d'un aliment (coupé ou cuit). Mais ces transformateurs doivent respecter des règles :

- Un aliment ne peut pas subir deux fois la même transformation. (Ex : une tomate ne peut pas être coupée deux fois.)
- Un plat composé ne peut pas subir de transformation. (Ex : on ne peut pas couper un plat composé d'une salade et d'une tomate.)
- Un transformateur ne peut recevoir un aliment s'il y en a déjà un sur celui-ci.

Un ajout plus fonctionnel est la création automatique des plats buts à partir du fichier texte de la carte.



Pour cela, nous avons séparé le fichier texte par un tiret "-". Cela permet de faire la distinction entre niveau et données de plats lorsque l'on lit le fichier texte.
En ce qui concerne les différentes conventions de nommage des plats, nous avions :

- Tomate : T
- Viande : V
- Salade : S
- Coupe : C
- Cuisson : R (roast)

La structure de lecture est séquentielle. Ce qui fait que la ligne "C R T" donne une "new Coupe(new Cuisson(new Tomate()))", soit une "Tomate - Cuite - Coupée". Pour ajouter un autre aliment au plat, il faut ajouter une virgule et continuer sur la même ligne. Il est également possible d'effectuer un saut de ligne et d'ajouter un nouveau plat en suivant la même convention de nommage.

Pour continuer le développement de l'agent artificiel, Julien a commencé par adapter l'heuristique pour lui permettre d'aller chercher autre chose que des tomates. Pour ce faire, nous avons eu l'idée d'utiliser l'algorithme UCS pour le faire. Le principe est de calculer un coût hypothétique pour la réalisation d'un plat en partant de la position initiale du joueur (selon l'état en cours) jusqu'au dépôt. Pour ce faire nous partons du nœud représentant le cas de départ et nous simulons des actions qui correspondent à chaque aliment nécessaire pour le plat. Chaque action est associée à des coordonnées (correspondant aux coordonnées de l'aliment). Puis nous faisons toutes les simulations nécessaires, réalisant les différentes

associations possibles en calculant le coût. Pour calculer le coût d'un "déplacement" vers un aliment, nous calculons la distance à vol d'oiseau entre le joueur avant déplacement et l'aliment, puis nous changeons les coordonnées du joueur par celle de l'aliment. Le joueur sera fictivement dans le mur (contenant l'aliment), mais cela ne pose pas de problème pour le coût puisqu'il y a l'action de prendre qui n'est pas prise en compte. Pour gérer les aliments à explorer et ceux restants, pour chaque état, nous disposons d'une liste d'aliments explorés. A chaque nœud, nous ajoutons l'action dans la liste d'aliments explorés.

Pour ce fonctionnement, nous avons rencontré plusieurs difficultés notamment liées au fait qu'il fallût prendre en compte beaucoup de choses. En effet, au départ, nous pouvons facilement générer le coût avec tout ce qu'il faut faire. Mais lorsque le jeu est à un stade "avancé", donc lorsque le joueur a déjà pris un aliment, l'état n'est plus le même et il faut bien faire attention à prendre en compte son inventaire. En effet, nous refaisons les calculs à chaque action. Parmi les problèmes, nous avons aussi eu la difficulté de les localiser. Nous produisons quelque chose qui, dans la théorie, est capable de fonctionner, pourtant cela n'était pas le cas. Julien a passé beaucoup de temps à localiser les différents problèmes et oublis faits par les autres personnes dans le reste du code (qui n'était pas lié directement à la partie artificielle).

Julien a fini son itération en réfléchissant à la prise en compte d'état pour les aliments et en commençant a développé des solutions pour ce point.

Thomas a quant à lui continué le développement de l'interface graphique en implémentant les nouvelles fonctionnalités décrites ci-dessus. Pour cela, il va créer une vue pour chaque type de bloc. Il a également créé des personnages différents en fonction du type de joueur sélectionné (violet pour AA, rose pour humain). Il va améliorer les entrées utilisateurs en passant par la console pour pouvoir choisir le type de joueur.

Thomas va de plus mettre en place la possibilité de pouvoir jouer à deux joueurs dans une seule partie. Ce système est une version primaire de la gestion de plusieurs joueurs en même temps. En effet, il faut attendre obligatoirement une interaction entre l'utilisateur et le clavier pour que le(s) agent(s) artificiel(s) joue(nt). Cette tâche sera automatisée dans les prochaines itérations.

Itération 3

Pour commencer l'itération, Julien a continué le développement de l’agent artificiel avec sa gestion actuelle en prenant en compte les états. Ainsi, l’agent artificiel était capable de prendre une tomate, la couper et la cuire puis d’aller chercher une salade. Pour ce faire, nous avons donc dû ajouter à l'heuristique la possibilité de détecter dans quel état doivent être les plats et d’ajouter le fait de passer par le transformateur associé avant de passer à la suite.

À ce moment, pour réduire le temps de processus, nous avons eu l’idée d’inverser le fonctionnement de l’agent artificiel. Avant, il fonctionnait en calculant le chemin complet du départ au dépôt en étant orienté par UCS qui retournait un coup pour le chemin estimé. Mais nous avons eu l’idée de commencer par estimer le chemin que prendra l’agent. Ce chemin estimé est une suite d'aliments avec leurs coordonnées (et les plaques de cuisson et planches). Puis on calcule, avec A*, le chemin vers le prochain aliment uniquement. Julien continuera donc de développer cet agent tandis que Hugolin et Nicolas continueront le premier.

À ce stade, la nouvelle version était capable de réagir globalement à l’autre utilisateur selon ses actions sur le terrain. En effet, comme nous calculons tout le chemin à chaque action, le joueur artificiel prendra en compte dans ses calculs les nouveaux éléments, et s’en servira, si besoin. Ce qui permet d’avoir un début d’agents qui collaborent. Les difficultés rencontrées sur ce point concernent notamment le fait qu’il faut prendre en compte beaucoup de points lors d’un ajout.

Concernant la première version de l’agent artificiel, Nicolas et Hugolin ont commencé à coder un système d’agent décentralisé, c’est-à-dire un agent qui prend en compte les intentions du joueur. Cette tâche a été entamée vers la moitié de l’itération, donc il a été question de trouver comment prendre en compte l’autre joueur en effectuant un brainstorming.
Nous avions noté que l’agent décentralisé devait avoir un temps d’analyse pour trouver l’action la plus optimisée pour chaque joueur, et, si l’action est par exemple de prendre une Tomate, la tâche serait supprimée de la liste de tâches de l’agent, pour éviter qu’il effectue la même tâche que les autres joueurs, et donc, qu’il soit à priori dans une optique de collaboration.

Dans cette itération, Nicolas a amélioré le système de recherche d’éléments. En effet, lors de l’itération 2, un algorithme de recherche de coordonnées a été mis en place pour l’agent artificiel pour qu’il puisse savoir où est quoi.
Par exemple, s’il cherche une Tomate, il pourrait trouver les coordonnées de chaque générateur de tomate et de chaque tomate sur la carte. Le problème est qu’à l’itération 2, nous avions implémenté cette fonction en effectuant une grande boucle switch en traitant chaque cas. Cela n’est pas un problème si nous cherchons les

coordonnées de tomates ou de générateurs, mais si nous cherchons quelque chose de plus précis (comme une Salade-Tomate-Coupée), nous devons ajouter beaucoup de cas dans la boucle avec beaucoup de conditions (instanceof par exemple), ce qui donne beaucoup de redondances et un grand nombre de ligne.
Pour alléger tout cela, il a fallu donner une String “type” à chaque attribut que l’on souhaite trouver et comparer ce string par rapport à ce qui est recherché. Cela est plus léger et automatisable, ce qui signifie que si l’on ajoute des blocs ou des recettes, la méthode n’a pas à changer.

Enfin, une dernière tâche mise en place est la meilleure intégration des plats but dans le fichier texte des niveaux. En effet, la structure de l’itération 2 n’était pas intuitive, avait quelques limites et beaucoup de contraintes (espaces à certains endroits et pas à d’autres). Nicolas a donc opté, en lien avec la nouvelle structure des aliments (état modélisé par un entier) une convention de nommage avec :
<nom de l’aliment> <entier de l’état>.
Donc, au lieu d’avoir “C R T” qui signifie “Tomate Cuite Coupée”, nous avons “T 3”.

Itération 4

Après l’itération 4, l’agent artificiel v2 (version de Julien) était capable de réaliser un plat avec un seul aliment avec état. L’un des premiers objectifs était donc de pouvoir ajouter une action “plan de travail”. En effet, si son prochain aliment à aller chercher est un aliment avec état, il doit poser son plat en cours sur un plan de travail avant de le continuer pour le récupérer par la suite. Pour ce faire, nous ajoutons une action de plan de travail puis le même plan de travail avant le dépôt pour bien y repasser.

C’est à ce moment que nous nous sommes rendu compte d’un problème dans la façon dont cela était géré. En effet, nous avions 3 statuts différents à prendre en compte pour gérer les données, 3 situations parmi lesquelles les traitements sont différents : lors de la situation initiale (tout le calcul à faire), lorsqu’il fait les actions et lorsque l’autre joueur change quelque chose alors que l’agent artificiel a quelque chose dans les mains.

Pour illustrer, nous prenons le cas d’un plat avec tomate cuite et coupée et d’une salade coupée. Le joueur prend la salade, la coupe, pose sur le plan de travail, prend la tomate, la coupe, la cuit, va chercher la salade coupée et dépose le tout. Cette situation génère une suite d’actions correspondant à ces dernières instructions. Or, lorsque l’agent a fini la salade et la pose, il ne faut pas qu’il la reprenne, il faut donc faire attention à cela. L’autre problème est le cas où l'agent artificiel à une tomate coupée dans les mains et que l’autre joueur pose une tomate

coupée et cuite quelque part. Cela bloquait l'agent artificiel qui ne savait pas quoi faire. Ces problèmes ont été provoqués par le fait que nous générons des données initialement à partir de DonneesJeu sans plus jamais faire appel à DonneesJeu. Donc nous modifions les données générées. Or, lorsque la partie se déroule, c'est DonneesJeu qui est modifié, donc il est différent du cas de départ et il faut générer les bonnes données (toujours dans UCS) et il faut le prendre en compte.

Pour résoudre ce problème, nous avons décidé d'utiliser les données de DonneesJeu au cours du processus de UCS. Nous ne chargeons plus les données au départ, mais chaque fois que nous cherchons une action à faire. Et chaque action faite se répercute sur DonneesJeu.

Parmi les objectifs de l'agent artificiel, nous retrouvons le fait qu'il puisse réagir aux actions de l'autre joueur sans se retrouver bloqué. Actuellement, il se servira de ce que fait le deuxième joueur. Mais si ce dernier bloque les planches à découper en posant des salades alors qu'il faut des tomates coupées, l'agent artificiel se retrouve bloqué. Désormais, il pose son plat en cours, libère une planche, et reprend le processus.

Hugolin et Thomas ont ajouté un nouveau type de joueur : l'automate. Ce joueur va recevoir à l'avance un ensemble d'instructions qu'il va réaliser sans prendre en compte l'état du jeu. Cela va être nécessaire notamment lorsque l'on voudra faire des mesures de calculs de temps des différents systèmes d'agents artificiels. Ce joueur est matérialisé par un rond jaune sur l'interface.

Nicolas a continué la mise en place de l'agent décentralisée, notamment en appliquant les idées trouvées lors du brainstorming de la troisième itération effectué avec Hugolin. Donc, lorsque l'autre joueur vise par exemple la tomate, l'agent décentralisé prend cela en compte et retire la tomate de sa liste de tâches, ce qui fait qu'il va chercher un autre aliment si le plat est composé de plusieurs aliments. Cette approche fonctionne relativement bien lorsqu'il est question de la préparation d'un plat composé de plusieurs aliments sans état. En effet, l'agent décentralisé voit que l'autre joueur se dirige vers un aliment, donc il se dirige vers l'autre aliment.

Thomas a créé un menu pour sélectionner les paramètres d'un niveau. Avant, le niveau choisi ainsi que sa durée étaient codés en dur et on devait renseigner les différents types d'AA dans la console. Désormais, tout passe par un menu en deux étapes. La première consiste à choisir le niveau souhaité parmi ceux disponibles. Une fois ce choix fait, un deuxième menu demande le type d'AA souhaité pour chaque joueur ainsi que le temps voulu pour la partie. Il ne reste plus qu'à lancer la partie pour prendre en compte tous ces paramètres.

Fonctionnement des agents artificiels :

Comme dit précédemment, nous disposons de deux fonctionnements différents pour les agents artificiels. La première (version continuée par Hugolin et Nicolas) est une version fonctionnant avec A* et utilise UCS pour son heuristique et la deuxième (dont le développement est fait par Julien) UCS puis A*.

Dans le premier cas, A* a pour objectif la résolution complète du niveau. Il calcule l'intégralité du chemin jusqu'à la résolution complète du niveau. Pour calculer le chemin, nous orientons les recherches à l'aide de l'heuristique calculée avec UCS. UCS (glouton), parcours toutes les possibilité d'actions (prendre une tomate, prendre une salade, cuire,…), teste différents ordres d'actions tout en calculant le cout de chaque action. Le coût est calculé en faisant la différence entre les coordonnées d'avant déplacement et après. L'avantage de cette version est qu'elle peut trouver une solution même pour une situation non prévue. De plus, elle trouvera la solution la plus optimale pour résoudre un problème.

Dans le second cas, nous utilisons UCS pour calculer le chemin global à faire. Nous obtenons l'ordre de chaque aliment et transformation avec les coordonnées de chacun, puis, nous utilisons A* pour calculer le chemin vers la première action. Cette version est plus rapide puisqu'elle exécute A* sur quelque chose de moins long, mais elle n'est pas forcément optimale puisque A* ne prend pas en compte la prochaine action à faire, donc elle ne se positionne pas au mieux. De plus, un autre problème est qu'il faut tout prévoir, elle ne fera pas une action non prévue. Par exemple, elle ne pouvait pas poser sur un plan de travail avant que ce ne soit prévu.

Le premier cas a été appliqué dans le cas d'un agent décentralisé. Ce dernier prend en compte les autres joueurs pour décider de sa prochaine action. En effet, avant que l'agent ne décide de son action, il va simuler l'action optimale pour les autres joueurs, puis décider quoi faire en fonction.
Prenons l'exemple où nous avons une salade et une tomate à assembler. Si le deuxième joueur prévoit d'aller chercher la tomate, car il est plus proche de la tomate que de la salade, l'agent décentralisé va retirer cet aliment de sa liste de tâches, ce qui fait qu'il va chercher la salade. Cela donne un aspect collaboratif, car l'agent "délègue" une tâche à l'autre joueur.
Pour le moment, l'aspect de collaboration, où un aliment est découpé puis donné à l'autre joueur pour optimiser ses déplacements n'a pas encore été implémenté.

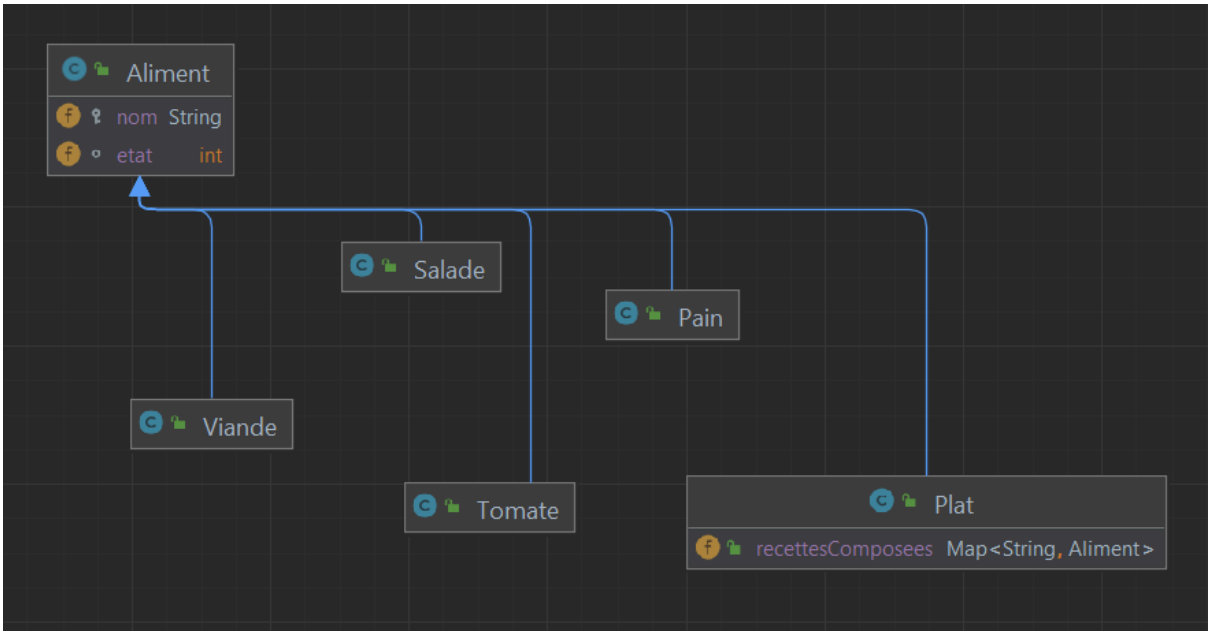
Dans les deux cas, les algorithmes sont capables de réagir aux actions de l'utilisateur puisque les calculs sont faits en prenant en compte toutes les informations présentent sur le niveau, et les calculs sont refaits à chaque début de tour. Donc si le deuxième joueur pose quelque chose qui servirait au premier, il sera capable de s'en servir.

En ce qui concerne les performances des deux versions de l'agent artificiel, nous avons des différences de temps. En effet, la première version de l'agent artificiel prend en moyenne 11 ms pour trouver une action, tandis que la deuxième version de l'agent prend en moyenne 2,5 ms. La différence est notamment due au fait que la première version de l'agent est décentralisée et doit prédire les actions de l'autre joueur, ce qui prend plus de temps, et c'est aussi dû au fait que la première version a un fonctionnement global plus lourd en général (calcule tout le chemin).

2. Présentation d'éléments dont nous sommes fiers

Nicolas Russo - Système de Plats

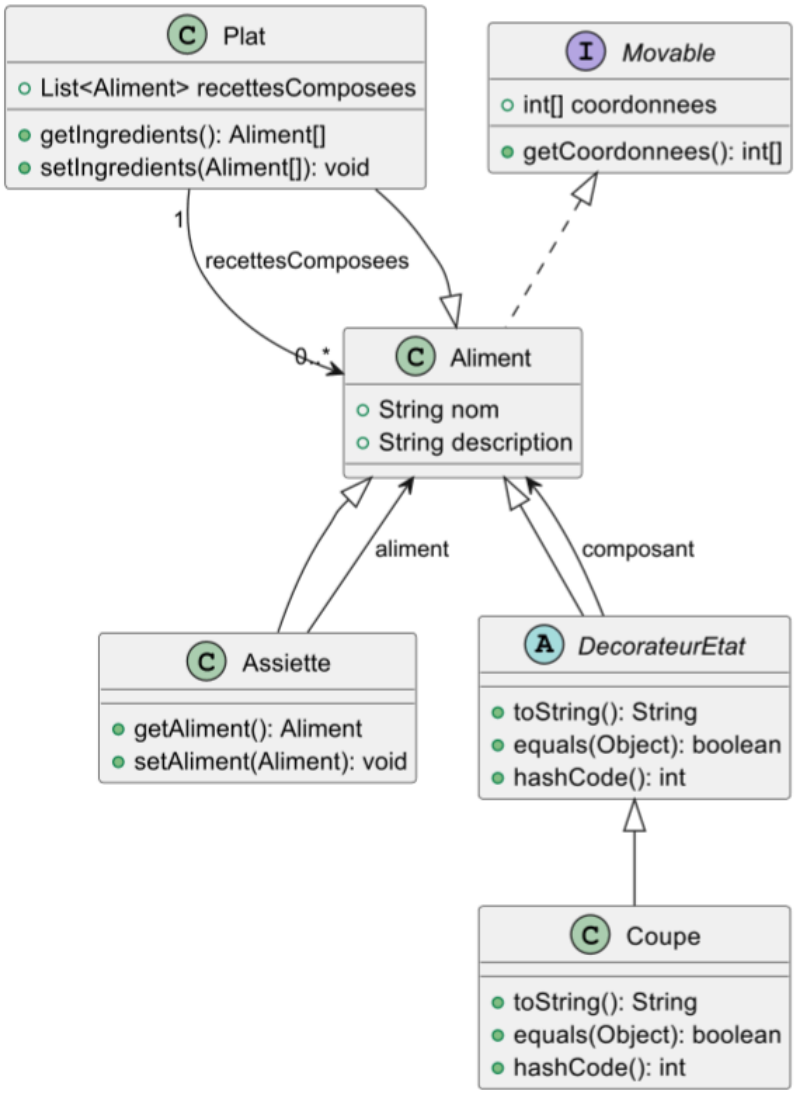
Un des principaux aspects autre que les agents artificiels qui ont été effectués sont la mise en place d'un système de plats. En effet, pour construire un plat à déposer sur le jeu Overcooked, il est requis de prendre plusieurs aliments, de les modifier (couper, cuire, …) puis de les assembler. Par exemple, pour former une "salade de tomates", il est requis de prendre une salade, de la couper, puis de l'assembler avec des tomates qui sont également coupées. Pour effectuer cela, il a fallu une certaine architecture de code pour que tout cela fonctionne correctement. Voici le diagramme de classe associé :



Ici, nous avons un Aliment qui possède une String nom (par exemple "Tomate") et un entier état. Pour faciliter le processus, des classes qui étendent d'Aliment ont été créées pour qu'elles soient utilisées dans la composition d'un plat.

En ce qui concerne les plats, une classe Plat a été créée. Cette dernière a une liste d'Aliments sous forme de Map, ce qui permet de relier un type d'aliment (comme "Tomate") à une valeur particulière. Cela permet d'éviter la duplication d'aliments de même type, qui est une action non autorisée sur le vrai jeu Overcooked.

Il était, au départ, prévu d'avoir un état à partir d'un patron décorateur, c'est-à-dire qu'une classe appelée Etat implémentant Aliment pouvait avoir comme composant un Aliment, ce qui permettait d'avoir plusieurs classes Etat qui composent un Aliment. Voici le diagramme associé :



Cette approche a été modifiée à partir de l'itération 2, car bien qu'elle coche toutes les cases, elle est très lourde pour notre manière de fonctionner. En effet, notre application va simuler chaque configuration possible d'une partie Overcooked sur une carte donnée, ce qui fait que chaque configuration aura son lot de plats, de modifications et de positions. Il est donc requis de copier chaque élément d'une partie pour lancer des simulations et éviter que l'on ait des effets de bord.
Cependant, avec le patron décorateur, il était fortement lourd et compliqué de copier un état et d'effectuer des vérifications d'égalité entre plusieurs aliments qui avaient des états similaires, mais arrangés différemment.

Après un entretien avec notre tuteur Vincent Thomas, nous avons opté pour une approche où nous avons un état qui est sous forme d'entier. En effet, nous ne prévoyons que trois états, qui sont "CRU", "CUIT" et "COUPÉ", ainsi que la fusion des deux derniers états qui donnent "CUIT ET COUPÉ". Cela peut être facilement modélisable par un entier qui va de 0 à 3, avec :

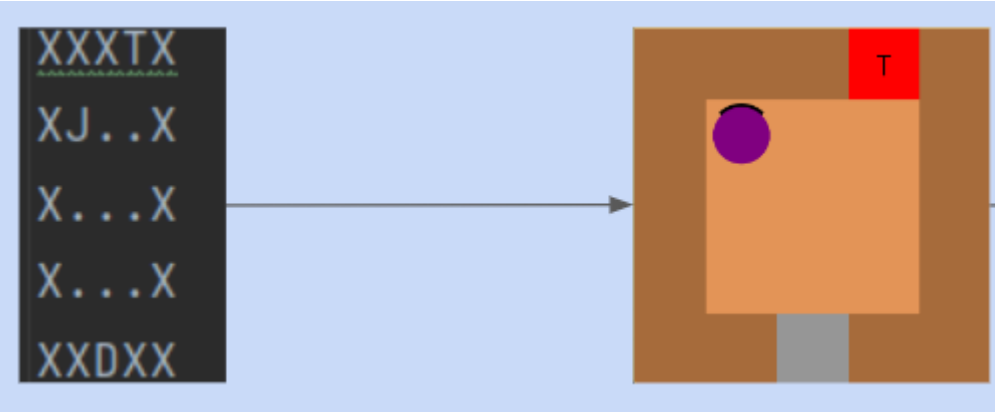
- 0 égal à CRU;

- 1 égal à CUIT;
- 2 égal à COUPE;
- 3 égal à CUIT et COUPÉ.

Cela permet de copier et de comparer les états facilement entre eux. Nous n'avons pas de volonté d'ajouter des états dans le futur (comme FRIT ou EMINCÉ), donc passer par un entier pour l'état est ce qui est de plus simple.

Hugolin Rouyer : Gestion du jeu de donnée

Le jeu de données est le cœur du projet. Sans lui, l'AA et l'interface graphique ne pourraient fonctionner. Le but est de transformer un fichier texte en un jeu de données. Pour cela, un algorithme va lire chaque caractère du fichier texte. Ce caractère correspond à un élément du jeu (plan de travail, joueur, transformateur…). Une fois ce caractère lu, il va créer l'élément dans le jeu de données et passé au suivant jusqu'à arriver à la fin de la carte. L'avantage de cette méthode est la création et la modification extrêmement simple.



Cependant, cette méthode possède un désavantage : les effets de bords. En effet, à chaque modification du jeu, on ne met pas à jour le fichier texte, mais directement le jeu de données. Sauf que le jeu de données est un objet et non une variable. Il faut donc à chaque fois recréer un objet. On va utiliser pour cela un constructeur par copie de l'objet. Cela est très important, notamment pour l'AA lorsqu'elle va créer des simulations et vérifier qu'une situation n'a pas déjà été explorée.

Le jeu de données va également permettre d'autres fonctionnalités :

- Vérifier qu'une action du joueur peut être réalisée au non
- Permettre la mise à jour de l'interface graphique
- Vérifier que deux instances sont identiques

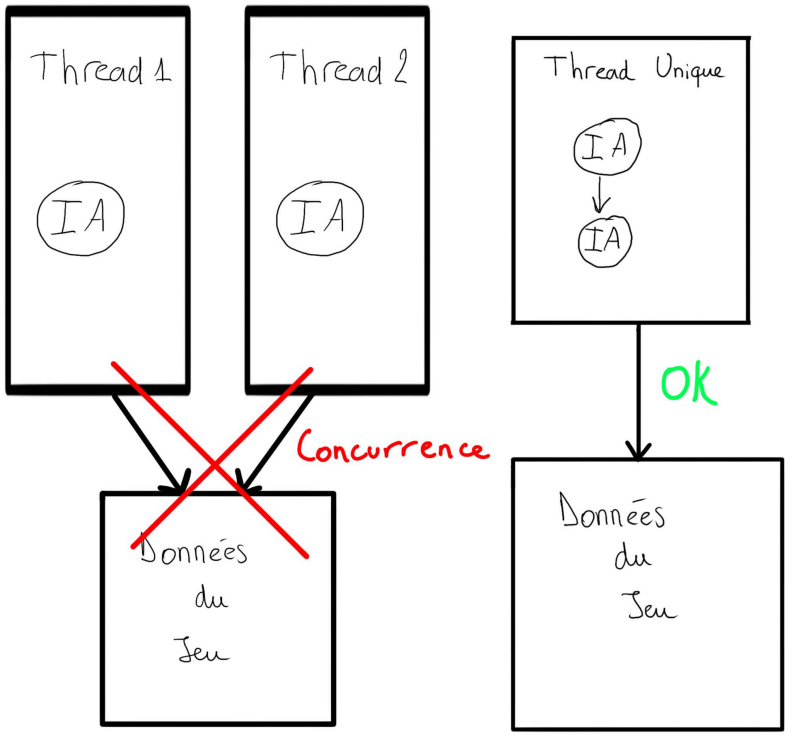
Pour s'assurer du bon fonctionnement de celui-ci, de nombreux tests unitaires ont été mis en place. Cela permet de vérifier si les effets de bords sont bien gérés, si

la création ou modifications d'éléments fonctionne, si la comparaison entre deux jeux de données fonctionne…

Cette partie n'a pas été aussi simple que prévu à la fois dans sa conception (des classes beaucoup trop lourdes) que dans la gestion des bugs. Beaucoup d'objets sont créés à partir de ce jeu de données et donc beaucoup de dépendances à gérer, notamment avec les effets de bords. Il a donc fallu plusieurs itérations pour que tous les bugs soient corrigés. D'autant plus que l'intégralité du code ne pouvait être testée à cause des trop grandes quantités de tests à rédiger. C'est pour cela que l'on a fait uniquement les tests sur les tâches critiques.

Thomas Robineau - Gestion de la concurrence des threads AA sur la structure de données

L'implémentation du tour par tour dans le jeu nécessite que les agents artificiels soient capables de choisir l'action qu'elles feront au prochain tour de manière indépendante, sans interférer avec le bon fonctionnement de l'interface graphique. La solution qui a été utilisée dans notre projet tutoré est l'utilisation de Threads au lancement de l'application, juste après la sélection des types d'AA.



Au lancement de l'itération correspondante, la première solution ci-dessus avait été mise en place, mais lors de l'exécution, les données du jeu semblaient incohérentes par rapport à ce qui était affiché à l'écran. Par exemple, quand 2 agents artificiels

voulaient interagir avec le même bloc / aliment. L'avantage de ce type d'implémentation est que tous les agents choisissent leur action de manière simultanée.

Pour pallier ce problème, il a fallu utiliser la 2e solution schématisée au-dessus. On utilise finalement un thread unique pour que les IA fassent le choix des actions de manière séquentielle. L'inconvénient est que le temps de réflexion des agents n'est pas optimal puisqu'ils doivent attendre que le précédent ait choisi son action.

De plus, l'écosystème JavaFX est un écosystème assez fermé, c'est-à-dire que dans notre cas, le thread était censé mettre à jour l'interface graphique de lui-même. Néanmoins, JavaFX pose une contrainte claire. Il est impossible pour les threads d'interférer avec une interface JavaFX. Pour remédier à cela, il a été obligatoire d'utiliser des AnimationTimer, afin que ce soit lui qui mette à jour l'interface, depuis le thread lui-même.

Julien : Développement des agents artificiels

Parmi les points essentiels du développement de notre application, se trouve le développement des agents artificiels. Cette tâche que j'ai réalisée durant l'intégralité des itérations me rend fière puisqu'elle aboutit à de bons résultats. Une raison supplémentaire qui fait que je suis fière de cette tâche est que je l'ai quasiment développé seul jusqu'à ce que Nicolas et Hugolin reprennent le développement de la première version pour la décentraliser. Les seuls points ou j'ai demandé de l'aide est concernant l'adaptation du jeu de données actuel. En effet, au cours du développement, j'ai dû demander aux autres membres de changer le fonctionnement de certaines parties pour que l'agent artificiel puisse fonctionner sans être trop ralenti. J'ai également demandé à de nombreuses reprises des méthodes pour accéder à certaines données ainsi que des corrections de bugs.
Constater après un long moment de développement qu'une fonctionnalité de l'agent artificiel fonctionne, par exemple, la dernière en date est de retirer des éléments des planches à découper si elles sont toutes occupées, est un point très satisfaisant.
Je ne détaillerais pas le fonctionnement de l'agent artificiel ici, puisque je l'ai déjà fait dans le reste du rapport, mais je suis très fière de l'aboutissement de ce dernier. Et également de constater que les deux fonctionnements que j'ai envisagés (A* avec UCS en heuristique et UCS puis A*) ont réussi. Aujourd'hui, je me rends compte que certaines choses auraient pu être mieux faites, mais cela me servira pour mieux penser les choses dans le futur.

3. Prévision pour les prochaines itérations

Le projet étant fondamentalement mis en place (interface graphique, plats, AA), les prochaines itérations ont pour but d'améliorer et d'optimiser l'ensemble du code. En effet, les agents artificiels sont fonctionnels, mais l'agent artificiel v1 (version actuellement décentralisée) a un temps d'exécution long et n'est pas encore optimal. Les deux prochaines itérations auront comme objectif d'améliorer ses performances et leur permettre de mieux collaborer. De plus, nous avons pour objectif de développer deux versions différentes des agents artificiels, correspondant aux deux versions déjà développées.

Nous souhaitons également complexifier les tâches à effectuer par les joueurs, donc en créant des cartes plus complexes et des plats plus compliqués, dans le but de tester les limites de l'agent artificiel (explosion combinatoire).
Après concertation avec notre tuteur, nous avons, également, prévu d'ajouter de meilleurs outils de calcul de performances, notamment en faisant collaborer les différents agents artificiels entre eux (AA V1 avec AA V2 par exemple) pour voir leur score (nombre de points, nombre de déplacements, …). Cela permettra de faire un comparatif clair et de se rendre compte des défauts et qualités de chaque algorithme de chaque AA.

Les deux prochaines itérations sont étendues sur 2 mois, ce qui nous permet en dehors des heures de projet de prendre plus de recul et de développer ce qui est le plus important.

4. Bilan

Nous sommes globalement fiers du travail que nous avons fourni. Nous avons globalement respecté le cahier des charges que nous avons réalisé initialement. À ce jour, les fonctionnalités suivantes ont été implémentés :

- Une interface graphique avec les informations essentiels,
- Des niveaux possédant des transformateurs et des plats complexes,
- Une première version d'agent artificiel décentralisé capable de prendre en compte l'intention de l'autre joueur mais ne peut faire que des recettes basiques (plats non transformés),
- Une deuxième version d'agent artificiel capable de faire des recettes beaucoup plus complexe mais ne prenant pas en compte l'intention de l'autre joueur,
- Deux agents artificiels capable de s'adapter aux actions de l'autre joueur,
- Un système de point et de temps pour une session de jeu,
- Un menu permettant de sélectionner les paramètres de jeux.

Les prochaines itérations vont se baser notamment sur l'amélioration des deux versions d'agents et comparer le plus performant par rapport à la qualité/temps. Nous espérons réussir à implémenter ces fonctionnalités et ainsi réussir à compléter les objectifs finaux du projet.