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. Introduction

A. 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.

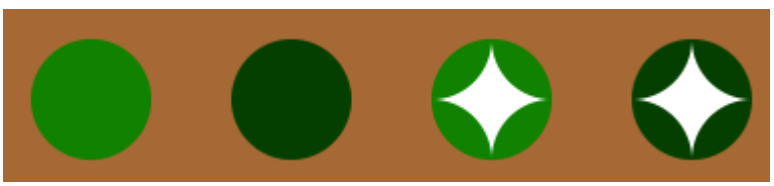
Ils peuvent suivre la progression du jeu grâce à une interface graphique contenant les informations nécessaires et une carte sous forme de grille représentant les différentes cases du terrain.

3. Une interface graphique

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

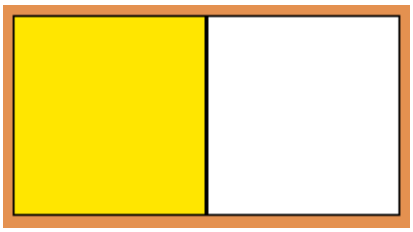
Un menu permettant la configuration de la partie, avec un sélecteur de niveau parmi une liste de niveaux préconstruits, un sélecteur de joueurs qui permet de choisir entre différents types d'Agent Artificiel (AA[1] V1[2], AA V2, Humain autonome) et un joueur humain. A noter que le nombre de joueurs maximum 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.

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.

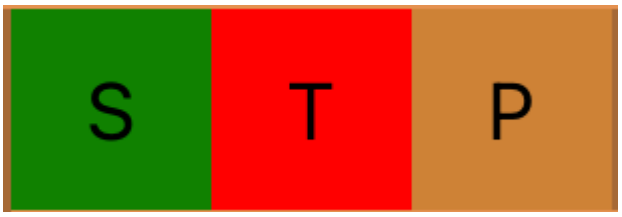


Aliments : Salade crue, Salade cuite, Salade coupée, Salade coupée et cuite

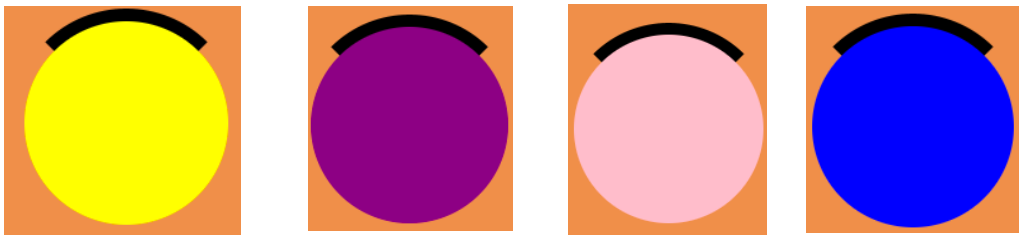
[1] AA: Agent Artificiel.
[2] V1: Les deux versions des agents artificiels seront définis par la suite.



Transformateurs : Plaque de cuisson, Planche à découper



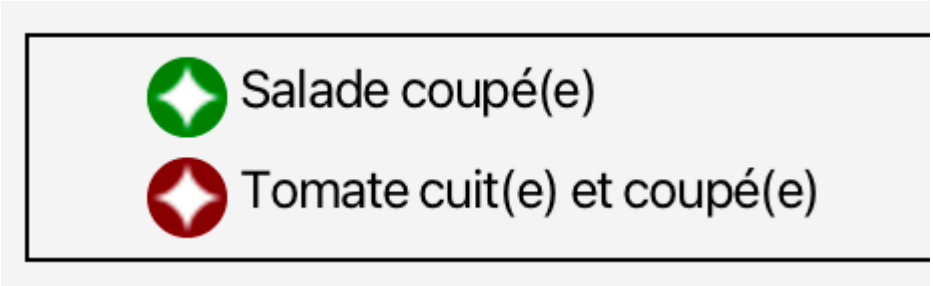
Générateurs : Salade, Tomate, Pain



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'interface va également afficher le prochain mouvement d'un agent. Ce déplacement est modélisé par un cercle noir.

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 (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 plusieurs agents artificiels avec des comportements transversaux et des utilisations d'algorithmes différents.

B. Présentation de l'équipe et des rôles

Globalement, chacun au sein du groupe a réussi à trouver sa place en fonction des prédispositions de chacun sur les différentes tâches à effectuer. Nous avons en résumé :

- Julien : Développement des IA / Supervision du projet,
- Hugolin : Création des tests / Développement des IA,
- Nicolas : Création du système de recettes / Développement des IA,
- Thomas : Création de l'interface graphique / Génération des statistiques.

C. Planning de déroulement des itérations

- Itération 1 :
    - Création de l'interface graphique,
    - Écriture d'une première IA sur des données fictives,
    - Mise en place de l'architecture du joueur,
    - Création de l'architecture visant à accueillir les fondations du projet,
    - Création du modèle du jeu,
    - Mise en place du système de recettes.
- Itération 2 :
    - Transformation de l'IA en utilisant le système du jeu,
    - Optimisation du système de recette,
    - Amélioration du jeu de données disponible,
    - Prise en charge des transformateurs (Planche à découper / Plaques de cuisson),
    - Prise en charge du tour par tour,
    - Création automatique des recettes à partir du fichier texte,
    - Optimisation des entrés utilisateurs claviers.

- Itération 3 :
    - Amélioration de l'IA de l'itération 2,
    - Nouvelle version d'une IA,
    - Commencement du système d'IA décentralisée,
    - Création d'une classe dédiée au calcul de temps,
    - Prise en charge du score et de l'ajout de points,
    - Meilleure intégration des plats but dans le fichier texte,
    - Automatisation du traitement des coordonnées des entités (Aliments, blocs, etc),
    - Complétion du tour par tour.
- Itération 4 :
    - Ajout d'un joueur automate pour niveau 0 et 4,
    - Mesure du temps de calcul des IA,
    - Ajout des menus pour choisir les types de joueurs et niveaux,
    - Amélioration de la 2e version de l'IA,
    - Première rédaction d'un README.
- Itération 5 :
    - Prise en compte de l'autre joueur dans l'IA et amélioration de cette dernière,
    - Ajout de nouvelles recettes / niveaux plus complexes.
- Itération 6 :
    - Exportation des stats au format CSV,
    - Optimisation des différentes IA,
    - Résolution de bugs / Optimisation.

II. Analyse

A. Découpage fonctionnel du projet

- Pôle Agents Artificiels :
  - Agent artificiel simple,
  - Agent artificiel semi-décentralisé,
  - Agent artificiel décentralisé basique
  - Agent artificiel décentralisé avancé
- Pôle Architecture
  - Génération automatique du modèle à partir du niveau,
  - Chargement des recettes à partir du fichier texte,
  - Duplication des états du modèle pour permettre les calculs des Agents Artificiels,
  - Implémentation d'un modèle de joueur global puis décliné par les Agents et humains,
  - Séparation de tous les éléments du jeu (Blocs, Aliments) en sous-catégories pour favoriser la précision lors de la génération du modèle,
  - Ajout d'une vérification de la composition des plats avant de valider l'ajout de points pour vérifier que le plat déposé est conforme à la recette du niveau.
- Pôle Interface Graphique :
  - Affichage des éléments du jeu (Blocs, Joueurs, Aliments),
  - Gestion du déplacement des joueurs via le clavier,
  - Intégration des Agents Artificiels,
  - Ajout d'un menu pour paramétrer les conditions de la partie.
- Pôle Statistiques :
  - Ajout d'un joueur automate pour simuler une présence humaine lors des statistiques,
  - Mesure selon le temps (5 secondes par défaut),
  - Intégration facile de nouvelles stratégies de mesures,
  - Exportation des données dans des fichiers séparés (Tours / Points / Niveaux).

B. Évolution par rapport au document préalable de décembre

Les différents objectifs évoqués durant l'étude préalable de décembre ont été respectés. Parmi ces objectifs, nous avons mis en place :

- un système de recettes complexe (transformation d'aliments),
- une interface simpliste,
- un sélecteur de niveau
- un système de point
- plusieurs types d'agents artificiels décentralisés différents
- un système de mesure des performances des agents.

Certains points n'ont pas été traités, comme l'ajout d'un timer pour la cuisson des aliments. L'ajout de cette fonctionnalité demande une grosse modification de l'heuristique. Or, nous n'avions pas prévu que cette fonctionnalité demande autant de changement dans le code. C'est pourquoi nous avons décidé de ne pas implémenter cette fonctionnalité. Bien évidemment, ce choix a également entraîné l'annulation de différentes fonctionnalités qui en découle comme la cuisson excessive d'un plat qui le rend inutilisable.

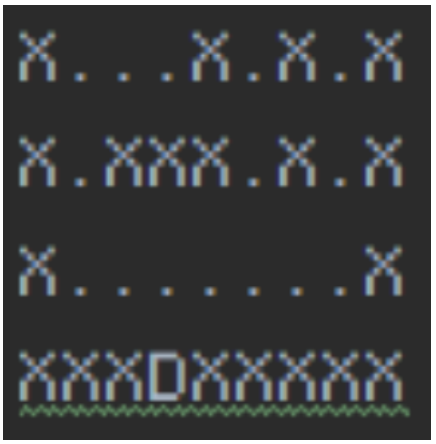
Initialement, il était prévu d'avoir des agents artificiels centralisés et décentralisés. Au cours des itérations, nous nous sommes rendu compte qu'il serait compliqué de réaliser les deux versions. Nous avons donc, après discussion avec notre Tuteur, décidé de ne faire que la version décentralisée.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

III. Réalisation

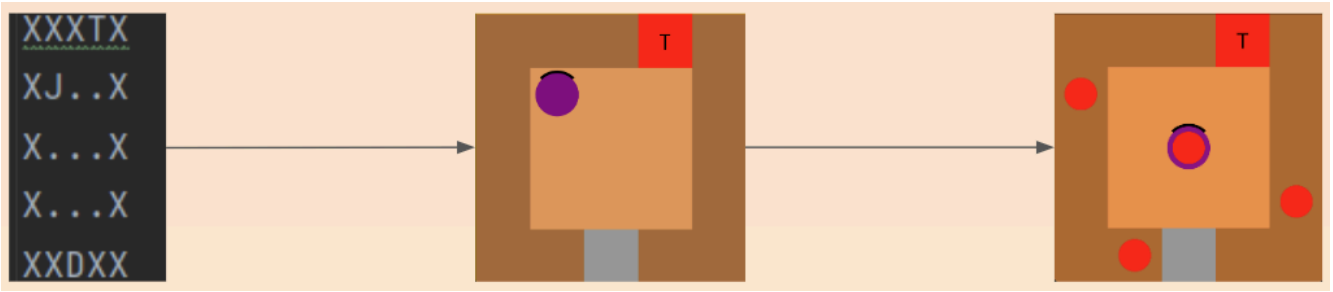
A. Fonctionnement du jeu de données

1. Création des niveaux

Pour la création des niveaux et pour les simulations de l'IA, nous avons mis en place un jeu de données. Un algorithme va transformer un fichier texte en jeu de données. Le fichier texte va être composé de caractères représentant chacun un élément du jeu (plan de travail, générateur…).



Extrait d'un fichier texte représentant un niveau. Le 'X' représente les murs et le 'D' le dépôt



Extrait de l'interface graphique après interprétation du jeu de données

Le jeu de données va également permettre à l'IA de savoir si une situation a été déjà explorée. Cela va prendre en compte la position de l'autre joueur, son orientation, les éléments sur la carte ainsi que leurs états. Les nœuds déjà explorés vont être retirés à l'arbre de décisions.

2. Système de recettes

Un élément fondamental liant le jeu et l'agent artificiel est le système de recettes. En effet, un plat est faisable par une recette et un système de plats but.
Nous avons au départ un aliment qui est un élément seul, comme une tomate ou une salade. Puis, nous pouvons lui changer son état, pour passer d'un état cru (0) à cuit (1), coupé (2) et cuit et coupé (3).

Nous pouvons également assembler différents aliments dans le but d'obtenir un plat.

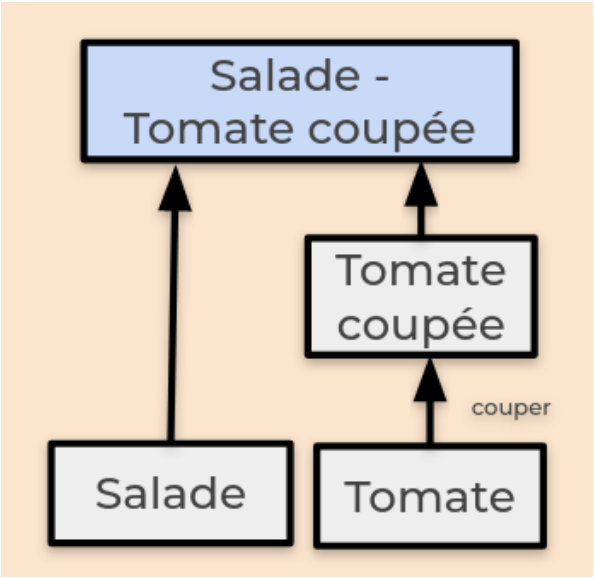


Schéma de composition d'un plat final, avec son acheminement

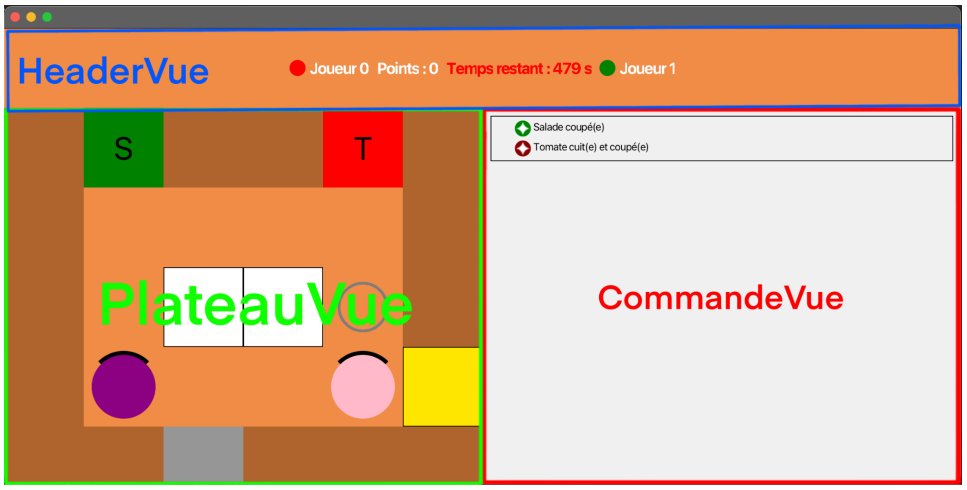
Nous voyons avec le schéma ci-dessus que pour effectuer un certain plat, il est nécessaire de prendre plusieurs aliments, effectuer des transformations (en passant par des éléments statiques du jeu comme des plaques de cuisson) pour ensuite les assembler.

Ce système a subi plusieurs changements durant le projet. Au départ, ce système fonctionnait avec le patron Décorateur pour l'ajout d'états sur un objet Aliment. Avec cela, il était possible d'avoir une Tomate - Coupée - Cuite - Sautée - Emincée - …, mais cela a été après coup simplifié, car la structure était difficilement clonable pour que l'agent artificiel puisse effectuer par la suite des simulations pour trouver un bon état but. Nous sommes donc passés d'un patron Décorateur à un entier "état" allant de 0 à 3. Cette structure primitive est bien plus facile à cloner.

B. Fonctionnement du jeu et de l'interface graphique

Le but de ce projet tutoré est de pouvoir projeter la représentation de l'interface graphique sur JavaFX, en essayant de respecter au maximum l'architecture MVC afin d'assurer la pérennité du code au maximum.

Afin de rendre les composants JavaFX utilisés dans l'interface graphique, nous avons fait le choix de séparer chaque composant dans une classe distincte, pour plus de praticité pour retrouver une logique en particulier.



Le premier calque se présente de la manière suivante. On retrouve une première série de vues pour les différents éléments globaux.

HeaderVue s'occupe de la gestion du temps et qui bloquera le jeu une fois que le compteur est terminé. De plus, cette vue s'occupe de la récupération des actions du tour en cours pour vérifier quel joueur a choisi son action, et donc afficher le cercle de la couleur correspondante.

CommandeVue est une vue statique qui récupère les recettes disponibles dans le niveau (stockées dans le modèle), et donc affiche les icônes correspondantes.

Enfin, la vue la plus complexe à réaliser est PlateauVue, puisque c'est celle ci qui va contenir toutes les petites vues des différents éléments comme les joueurs, les aliments, les blocs, etc.

C'est à ce moment-là qu'intervient le 2e calque de vues, donc pour les différents éléments qui vont se déplacer au fil de la partie. À savoir, que chaque élément de couleur différente correspond donc à une vue différente.

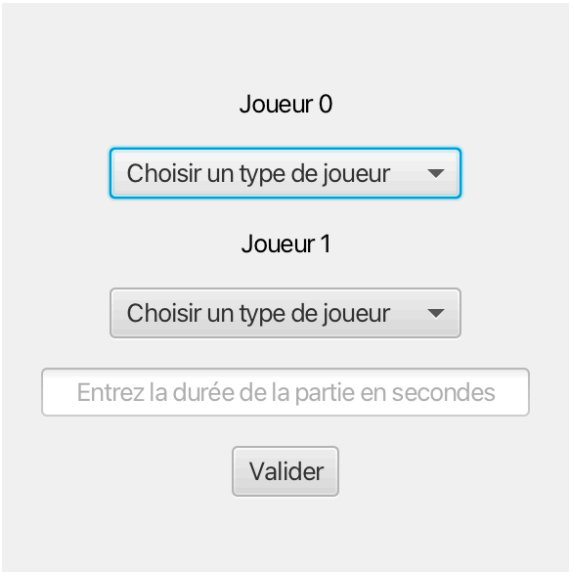
De plus, en relation avec le tuteur, il nous a été demandé de pouvoir afficher l'intention de l'agent artificiel sur la carte, semblable à une prédiction. Cet ajout a été réalisé en projetant un cercle de couleur grise.

Concernant la gestion des entrés claviers faîtes par le joueur humain, nous avons créé un contrôleur dédié à ça qui est chargé, une fois que l'humain a appuyé sur une touche, de vérifier que l'action qu'il entreprend de faire est légale, c'est-à-dire qu'elle fasse partie de l'espace d'actions qui lui est associé (notamment pour éviter que le joueur rentre dans un mur, etc). Dans la théorie, dès qu'un joueur (IA ou humain) choisit une action, elle devrait en théorie être visible immédiatement sur l'interface graphique. Mais ce n'est pas le cas.

Comme notre application implémente un système de tour par tour, il a fallu créer un mécanisme de stockage des actions choisies par les joueurs dans le tour en cours. C'est ici que la Classe ActionsDuTour entre en jeu. On a simplement une table clé - valeur, utilisant comme clé le joueur, et comme valeur, l'action que ce joueur a choisi.

À chaque ajout à cette table, elle vérifie si le nombre de joueurs de la partie est le même que le nombre d'actions dans la table. Si c'est le cas, alors ActionsDuTour se charge de résoudre les actions et de les afficher sur l'écran, et de passer au tour suivant simplement en effaçant le contenu de la table. Afin que l'IA "patiente" le temps que l'humain (ou l'autre joueur) choisisse son action, en la faisant attendre via des logiques internes à Java.

Auparavant, il était nécessaire de rentrer directement le niveau et le(s) type(s) de joueur(s) que l'on voulait utiliser dans les arguments de lancement du programme. Ce système peu ergonomique a été remplacé au profit de plusieurs interfaces qui mènent sur l'écran principal de l'application (présenté plus haut).



La première interface permet de choisir le niveau que l'on souhaite utiliser pour la nouvelle partie, via un menu déroulant qui présente les niveaux que nous avons déjà créés (bien que l'utilisateur puisse créer soi-même ses propres niveaux via la liste de caractères associée).

Une fois le niveau choisi, l'application va détecter le nombre de joueurs présents sur la carte (via le nombre de caractères J présents sur le fichier texte) et afficher le nombre de champs requis pour sélectionner le type de joueur que l'on souhaite utiliser pour la partie.

La potentielle difficulté rencontrée, qui a demandé le plus de travail, est l'intégration des différentes IA à l'interface graphique pour les faire travailler ensemble. Pour ce faire, une fois que le jeu est réellement lancé, le mécanisme d'IA est lancé dans un thread à part de l'interface tout en les liant un minimum en incluant certains éléments de l'interface graphique comme le plateau, et bien évidemment le modèle.

Concrètement, le scénario du thread est exécuté tant que la partie n'est pas terminée (que le temps imparti n'est pas écoulé). On fait choisir une action à l'IA, et on l'ajoute aux actions du tour présentées plus haut. Vient ensuite le mécanisme d'attente pour permettre à l'autre joueur de choisir son action, puis de la mise à jour de l'interface graphique. À savoir que la mise à jour est possible aussi bien ici, que quand la partie comporte 2 humains.

Étant dépendants du langage de programmation et plus particulièrement de JavaFX, qui possède une logique est des restrictions qui lui sont propres, il n'est pas possible de modifier un quelconque composant de JavaFX directement puis un thread. Cet aspect de JavaFX est particulièrement problématique puisqu'il est impératif que le thread de l'IA soit en mesure de mettre à jour l'interface graphique puisqu'il n'est pas obligatoire qu'il y ait un humain dans la partie qui puisse s'occuper de ce rôle. Nous avons trouvé un moyen de contourner cette restriction via l'utilisation d'un timer qui met à jour l'interface graphique en interne et non plus via le fameux thread.

C. Fonctionnement des agents artificiels

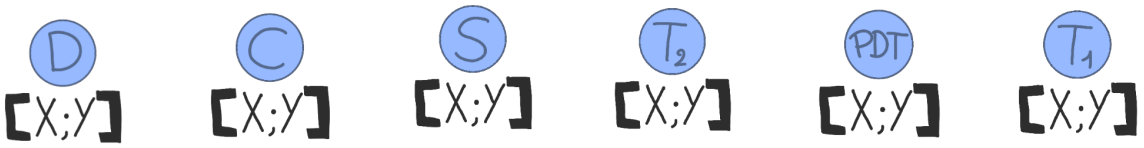
L'objectif des agents artificiels est de trouver le meilleur chemin pour réaliser un plat dans un temps imparti. En effet, on souhaite que les agents puissent résoudre le niveau sans que l'expérience utilisateur soit affectée, puisque l'on demande à l'agent artificiel de calculer une action à chaque tour de jeu. Pour ce faire, on associe un plat but à un niveau, qui sera découpé pour pouvoir le résoudre en se servant des éléments présents sur le niveau avec leurs coordonnées.

Pour résoudre ce problème, nous avons réalisé différentes versions d'agents artificiels utilisant des sous-algorithmes avec chacun leurs spécificités. Le premier est l'algorithme de recherche d'intention fonctionnant avec UCS et le second est l'algorithme de recherche de chemin fonctionnant avec A*.

1. Présentation des sous-algorithmes

Le fonctionnement des deux algorithmes A* et UCS est assez similaire. Ils créent tous deux un graphe en partant d'un état initial et en y appliquant une action pour obtenir un nouvel état. La différence entre ces deux algorithmes est qu'A* prend en compte l'heuristique d'un État. En effet, UCS explore en premier les nœuds dont le coup **passé** (coût entre un nœud et le nœud d'origine) est le plus faible. Alors qu'A* prend également en compte le **coût estimé** jusqu'au nœud final, appelé **heuristique**.

L'algorithme de recherche d'intention fonctionne avec comme espace d'action, les différents éléments de notre niveau auquel on associe leurs coordonnées telles que :



Légende:
D : découpe; C : cuisson; S : Salade; T : Tomate; PDT : plan de travail.

Ainsi, en réalisant les bonnes combinaisons d'actions, l'algorithme peut fournir une liste de directives estimées qu'un agent fera. Pour ce faire, il regarde le plat but et simule les actions dans l'optique de reproduire le plat but. Pour réaliser l'exploration des nœuds, l'algorithme explore, donc, les nœuds dont le coût passé est le plus court en premier. Ce coût est calculé en réalisant la différence entre les coordonnées d'un nœud (de l'action qui permet d'y accéder) et la coordonnée du

nœud d'avant. Les coordonnées du premier nœud correspondent aux coordonnées du joueur. On considère donc que le joueur se déplace jusqu'aux coordonnées de l'action suivante en ne se préoccupant pas des murs. Nous avons réalisé différentes versions de cet algorithme pour essayer de prendre en compte certains points, ou non.

La première version, qui est aussi la version que nous avons conservée, réalise un tri dans les actions à réaliser. En effet, à partir d'un état, nous disposons d'un assez grand nombre d'actions possibles, dont certaines ne sont pas forcément pertinentes. Par exemple, si nous cherchons à obtenir une tomate coupée et que nous avons une tomate dans les mains, il n'est pas pertinent d'aller poser la tomate. En procédant à ce tri dans les actions, nous sélectionnons les branches que l'algorithme va devoir explorer. Ceci est pertinent puisque sectionner des branches tôt dans l'arbre réduit grandement la durée de l'algorithme puisque ces "grandes branches" ne seront pas explorées.

Pour réaliser ce tri, on réalise une comparaison entre les éléments présents sur le niveau, l'élément dans l'inventaire du joueur et le plat but. En ne conservant que l'action qui avance le plus vers le plat but, nous réduisons ainsi les nœuds à explorer. La création de ce tri a été quelque chose de compliqué puisqu'il y avait de nombreux cas à prévoir, tel que le cas où un aliment dans un état plus avancé est présent ou le cas où l'agent a besoin de poser ce qu'il a dans ses mains pour traiter un autre aliment.

Le problème de cette version est que la suite d'instruction retournée n'est peut-être pas optimale. En effet, en partant du prédicat qu'il y a une tomate coupée (qui est le plat but) à l'autre bout du niveau, il serait peut-être plus rapide de prendre une nouvelle tomate et de la couper. C'est ici le principe de la seconde version. En effet, en ne réalisant pas cette sélection de nœuds, nous obtiendrons la meilleure solution possible. Cependant, cette nouvelle version a été abandonnée, car beaucoup trop lente à s'exécuter. La première version s'oriente directement vers le bon objectif contrairement à la seconde qui réalise de nombreuses bifurcations (ex : prendre une tomate, la poser à côté, puis une autre tomate, etc) et boucles (prendre puis poser), qui sont des actions qui seront explorées en premier (cas des bifurcations), car moins coûteuse que de traverser le niveau, par exemple.

L'algorithme de recherche d'intention prend partiellement en compte les actions de l'autre joueur. Par exemple, si ce dernier pose une salade coupée quelque part et que le premier joueur en a besoin, la salade coupée sera prise en compte dans les nouveaux calculs (en rappelant que nous réalisons les calculs à chaque tour). L'effet de l'autre joueur sur le terrain est donc pris en compte, tout comme l'inventaire de ce dernier.

La version conservée présente donc le problème de ne pas être optimal puisqu'elle ne trouve pas forcément le meilleur chemin. Une solution d'amélioration serait, probablement, de fonctionner avec un algorithme A*, donc en prenant en compte l'heuristique. En effet, cela permettrait de s'orienter vers les nœuds qui avancent le plus la situation et non pas seulement ceux qui coûtent le moins. Par manque de temps, nous n'avons pas eu le temps de l'implémenter, mais si le projet était amené à continuer, ce serait le prochain point qui serait travaillé.

L'algorithme de recherche de chemin fonctionne avec comme espace d'action, les différentes actions directement disponible dans le jeu, soit :

Droite - Gauche - Haut - Bas - Prendre - Poser - Découper - Cuire

L'objectif de cet algorithme est de calculer une suite d'instruction directement applicable sur le terrain dans l'optique d'atteindre un objectif. Cet algorithme simule les actions et déplacements directement sur une version copiée de l'instance de jeu en cours. Il simule les déplacements comme s'ils avaient lieu et génère un graphe avec une action sur chaque branche et un état de jeu, sur lequel a été appliqué l'action, à chaque nœud. Lors de l'exploration du graphe, l'algorithme traite en premier ceux ayant un coût passé et une heuristique le plus faible. Étant donné que cet algorithme calcul un chemin entre un point A (départ) et un point B (objectif), l'heuristique correspond à la distance (sans prendre compte des murs) entre les coordonnées à un instant T et le point B.

Avant d'avoir réalisé cette version de l'algorithme de calcul de chemin, nous avons commencé par créer une version utilisant A* sur le jeu complet et se servant de l'algorithme de calcul d'intention pour obtenir l'heuristique nécessaire à A*. En effet, l'algorithme de recherche d'intention retourne une liste d'intention avec leurs coordonnées associées. Nous prenions donc le coût de cette solution pour heuristique. Le problème de cette solution est qu'utiliser A* sur le jeu complet, donc en partant de la situation initiale jusqu'au plat but, et non pas un objectif intermédiaire, n'est pas viable puisque trop lent. Pour des plats complexes, cela générait trop de nœuds. S'ajoute à cela le fait que l'appel de l'algorithme de recherche d'intention se faisait trop de fois. Cette version a donc été abandonnée au profit de la version mentionnée précédemment. Ces sous-algorithmes sont donc utilisés dans des algorithmes plus globaux pour résoudre les niveaux.

2. Présentation des différents algorithmes

Pour résoudre le niveau dans sa totalité, nous avons implémenté plusieurs algorithmes utilisant les sous-algorithmes précédents. En les utilisant dans certains ordres ou en réalisant des traitements supplémentaires, nous obtenons différents résultats.

La version principale que nous avons développée est l'agent artificiel semi-décentralisé. Cet agent est capable de résoudre un niveau plutôt efficacement sans temps de calcul. En effet, ce dernier calcule, dans un premier temps, le prochain objectif d'un joueur, à l'aide de l'algorithme de calcul d'intention. Une fois la liste d'intention récupérée, on définit la première intention comme objectif de l'algorithme de calcul de chemin. Cette version est optimale, car elle calcule le moins de choses possible. L'algorithme le plus coûteux étant celui de recherche de chemin, on limite son nombre d'actions calculé en ne lui donnant qu'un sous-objectif. Cette version est directement tirée d'une version 0 qui ne prenait pas en compte l'autre joueur. Nous avons également réalisé plusieurs agents décentralisés, qui prennent donc en compte les intentions des autres joueurs par plusieurs moyens.

Nous avons réalisé une première version qui simulait l'intention de l'autre joueur, afin de déterminer ce qu'il allait faire. En conséquence, l'agent allait effectuer autre chose, en retirant l'élément que l'autre joueur va faire du plat but. Par exemple, si l'autre joueur va chercher une tomate, l'agent va supprimer la tomate de son plat but à lui, ce qui va lui permettre de modéliser ses recherches sur le reste du plat but. Cette version est fonctionnelle, et nous avons bien remarqué des capacités à collaborer, notamment en effectuant une autre tâche que l'autre joueur (chercher une salade pendant que l'autre joueur cherche une tomate). Cependant, cette version est très lente, demande beaucoup de temps de calcul par rapport aux versions suivantes, bien qu'elle soit fonctionnelle, car elle répond au besoin. L'agent prend en moyenne 11 millisecondes pour trouver une action.

Ensuite, nous avons réalisé une deuxième version, plus optimale et plus rapide, permettant de prendre en compte l'intention des autres joueurs en fonction de leur inventaire. Donc, si l'autre joueur porte une tomate, l'agent ne va pas chercher une autre tomate. L'agent prend en moyenne 2,5 millisecondes pour trouver une action.

Vers la fin du projet, nous avons fait une troisième version combinant les deux premières, donc en utilisant la manière de prédire les actions des autres joueurs de la V1 et la rapidité de la V2. Cette version est plus rapide, mais n'a pas pu être assez travaillée pour être performante, comme le montreront des statistiques.

3. Difficultés rencontrées

Nous avons rencontré de nombreuses difficultés dans le développement des agents artificiels :

- Complexité des décisions à chaque tour :

L'une des principales difficultés a été de permettre aux agents artificiels de prendre des décisions complexes en un temps rapide, ce qui est crucial dans un jeu comme Overcooked. Les agents doivent non seulement décider des actions à entreprendre, mais aussi optimiser leur comportement en fonction de l'état du jeu, du plat but et des obstacles présents sur la carte.

- Optimisation des Algorithmes :

Développer des algorithmes efficaces pour que les agents prennent des décisions rapides et pertinentes a été un défi majeur. Cela implique de trouver le bon équilibre entre la précision des décisions et la vitesse d'exécution. Nous avons dû explorer différentes approches algorithmiques, telles que les algorithmes de recherche heuristiques ou évolutifs, pour parvenir à des résultats satisfaisants.

- Équilibrage entre rapidité et qualité des décisions :

Un défi crucial a été de trouver le bon compromis entre la rapidité des décisions prises par les agents et la qualité de ces décisions. Par exemple, simplifier les modèles de décision pour accélérer le temps de réponse peut affecter négativement la précision des actions des agents.

- Optimisation du système de plat

Pour que l'agent puisse effectuer ses simulations, il a fallu trouver un système de plat qui soit assez simple à cloner, mais aussi assez étendable pour y ajouter des variétés d'états (par exemple, cuire et couper un aliment). Le défi a donc été de trouver le juste milieu entre optimisation et possibilités d'extension avec ce système. Cela a été trouvé au fur et à mesure du projet.

- Gestion de la coordination entre agents :

Coordonner efficacement les actions entre plusieurs agents artificiels a également été une difficulté. En effet, les agents doivent coopérer pour atteindre des objectifs communs.

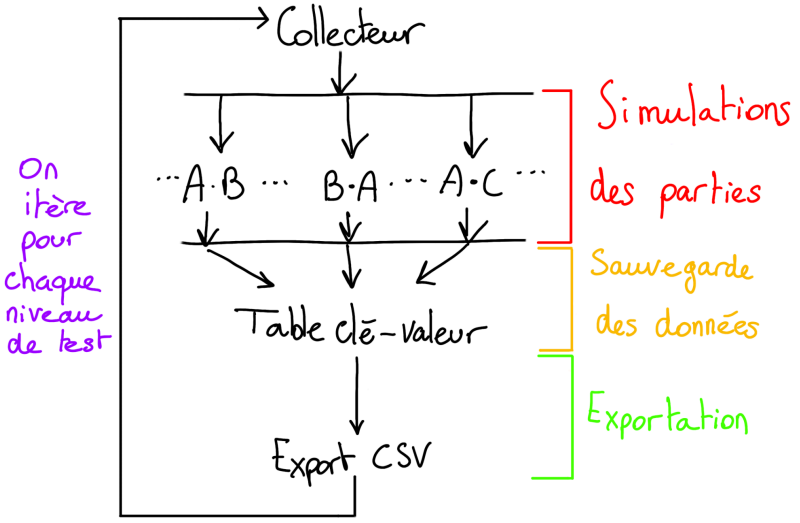
En résumé, les principales difficultés liées aux agents artificiels ont été de produire un algorithme trouvant la solution la plus juste possible, dans un temps rapide. Nous avons dû réfléchir activement à comment optimiser nos processus et comment agencer nos différents algorithmes pour obtenir un résultat concluant. Il nous a également fallu réfléchir à certains compromis pour accélérer les processus en ne réduisant pas trop la qualité des résultats fournis.

D. Génération de statistiques

Afin de pouvoir présenter les résultats des différents agents artificiels de manière concise, nous avons développé un programme permettant de simuler des parties (sans interface graphique puisque c'est inutile dans cet environnement) avec tous les types d'IA en joueur 1 et en joueur 2.

Le programme lance les simulations de manière séquentielle, c'est-à-dire qu'on lance les parties les unes après les autres. À chaque simulation, on stocke le nombre de tours et de points effectués par le Duo dans une table clé valeur de structure **{Duo d'IA ⇒ Points / Tours}** afin de minimiser la concurrence sur le fichier qui sera le tableau CSV.

Pour avoir des résultats ayant du sens, on lance les simulations 3 fois à la suite et on garde la valeur médiane des 3, et c'est celle-là qui sera enregistrée dans le fichier.



Par exemple, une fois que le programme a terminé de tourner, le tableau à double entrée généré ressemble à cela :

Points/5s	IA	IADecentrV2	IADecentrV3	Automate
IA	49568	49538	15078	81822
IADecentrV2	49554	49750	15104	81789
IADecentrV3	17372	17412	9532	23410
Automate	82838	82849	20805	-1

Correspondance des IA:

- IA: Version d'origine des Agents artificiels (V0) ne prenant pas en compte l'autre joueur mais réagissant à ses actions sur le jeu;
- IADecentrV2: Agent artificiel semi-décentralisé;
- IADecentrV3: Agent artificiel décentralisé avancé.

À savoir que l'on a également créé des joueurs automates pour que les IA puissent essayer de jouer avec une simulation de présence humaine. Ces automates ont des actions prédéfinies inscrites en dur dans le code. De plus, on ne simule pas une partie avec 2 automates puisque cela ne met pas en scène d'IA, d'où le résultat -1.

Ici, le tableau permet de regarder quel type d'IA est le plus robuste, en présence de quelle IA. Dans notre cas, ce sont les types IA et IA Décentralisée V2 qui semblent les plus robustes avec des résultats très similaires. En revanche, l'IA Décentralisée V3 a des performances un peu en deçà des autres dus au manque de temps pour perfectionner l'algorithme.

IV. Bilan

De notre point de vue, le projet est une réussite. Nous avons réussi à implémenter les tâches critiques. Nous avons une version stable du logiciel (pas de bug critique). Voici la liste des fonctionnalités implémentées :

- Une interface graphique avec les informations essentielles,
- 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,
- Une troisième version d'agent artificiel prenant en compte toutes les intentions de l'autre joueur et les retire de sa propre liste d'intentions (ce qui donne une délégation de tâches),
- Deux agents artificiels capables 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.
- Un système de mesures des statistiques des différents agents

Notre projet ne pourrait selon nous pas être repris l'année prochaine, étant donné qu'une grande partie de recherche est implémentée et que certaines modifications reviendraient à reprendre le projet de 0. Nous nous sommes focalisés sur un type d'IA pour le projet, et d'autres types peuvent être tout aussi adéquats. Donc, si le sujet est repris, ce rapport peut être une piste de recherche pour trouver une solution sous un autre angle. Les résultats de ce rapport peuvent également être utilisés à titre comparatif.

V. Annexe

Le projet est effectué en JavaFX avec les dépendances Maven. Le projet est à lancer depuis le logiciel IntelliJ IDEA. Pour exécuter le projet, la classe Java à lancer est **OvercookedJavaFX** située à la racine du dossier de développement JavaFX (“src/com.[...]/java/”). À partir de cette classe, l’ensemble des procédures (choix de niveau, choix des personnages, choix du temps) sont directement spécifiés sur l’interface graphique, et plusieurs choix sont possibles en fonction des besoins (jeu entre deux AA, jeu Humain/AA…).