BERNARD - ROBINEAU - ROUYER - RUSSO

Etude préalable

Sujet 2 : Collaboration entre agents artificiels

Description du Projet

L'objectif principal de ce projet est de concevoir un environnement de simulation inspiré du jeu "Overcooked", dans lequel plusieurs agents artificiels 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'envoie 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 mets dans un environnement difficultueux 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.
Nous devons donc réaliser le développement d'un simulateur de jeu qui reflète le concept de ce jeu. L'environnement simulé doit comprendre des éléments tels que des ingrédients, des plats à préparer, une carte avec les chemins pour y accéder, des planches à découper, des assiettes, des poêles pour cuire, ...
L'application sera réalisée en java et comportera une interface graphique JavaFX. Avec la possibilité de jouer à plusieurs joueurs. L'application devra intégrer des agents artificiels indépendants ou non. 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

Le jeu nécessitera une collaboration entre différents joueurs et fonctionnera en tour par tour. Les joueurs joueront leur tour respectifs en même temps dans l'optique de réaliser un ou plusieurs objectifs (les plats). Il y aura donc la nécessité, pour les agents artificiels, de pouvoir coordonner leurs actions avec les autres joueurs. Les joueurs seront donc soit humains, avec des contrôleurs divers, soit artificiels. Ils pourront 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 devra contenir un ensemble d’informations permettant le bon déroulement du jeu. Ces informations seront :

- Un menu de jeu permettant de:
    - Choisir le niveau,
    - Choisir la configuration des joueurs (humain ou IA).
- La grille correspondant aux cases du terrain,
- Des objets fixes :
    - Les cases pour la cuisson,
    - Les cases pour la découpe,
    - Les cases d'apparition des ingrédients,
    - La case de dépôt du plat.
- Des objets déplaçables :
    - Les ingrédients (et les ingrédients composés),
    - Les ustensiles de cuisine.
- L’emplacement des joueurs,
- Les limites du terrain,
- Les informations nécessaires au jeu :
    - Les plats à réaliser,
    - Un score,
    - Un compte à rebours.

L’ensemble de ces informations sera présente dans un fichier texte. L’application comportera un système de lecture du fichier qui sera chargé de lire le fichier et d'initialiser les données pour l’affichage et l'exécution de la partie. Il faudra prêter attention à la gestion des données. Les informations devant être présente sur ce fichier sont donc:

- fichier texte
    - Points de départs Joueurs (premier joueur est le premier sur la carte dans l’ordre de lecture)
    - plan de travail (limite du terrain)
    - lieu où apparaissent les aliments
    - lieu pour cuire
    - lieu pour découper
    - lieu pour déposer
    - liste des plats du niveau
        - Nom du plat, nombre de points, tour d’apparition, nombre de tour de durée.

4. Déroulement de la partie

Le jeu débutera avec l'initialisation du niveau qui sera suivie de la demande d'action aux joueurs. Les calculs liés à l'agent artificiel se feront à chaque début de tour puisqu'il doit réaliser une action à ce moment. Les actions possibles sont donc : un mouvement, une action (ex: prendre ou poser un objet, découper,...). Une fois que toutes les actions des joueurs sont reçues, l'application réalise les actions de tous les joueurs ainsi que leur traitement associé. Il faudra prêter attention à la gestion du cas où 2 joueurs font la même action (**Priorité au premier joueur ayant réalisé l'action**). Le niveau se terminera selon un seuil défini, soit un temps, soit un nombre de points, soit un nombre de plats.

5. Multijoueur à 2 Joueurs

Au cours du développement de l'application, il faudra réaliser plusieurs étapes, parmi celles-ci, il y aura la création d'un système multijoueur (à 2 joueurs dans un premier temps). Cette étape implique l'extension du jeu pour permettre le multijoueur avec deux joueurs. Il faudra faire attention à la source des joueurs. Les joueurs seront soit humains, soit artificiels.

6. Intégration d'un Agent intelligent

Intégrer un agent artificiel nécessite un calcul de chemin, et une gestion des chemins, nous devons gérer les plats et leurs étapes, ainsi que la réalisation du calcul en fonction des besoins ET de l'état actuel des choses.

7. Intégration d'un 2ème Agent intelligent avec cerveau central

L'intégration d'un 2ème agent artificielle lié au premier permettra au cerveau de disposer de deux actions possibles pour la réalisation des tâches. Il aura donc plus de marge de manœuvre.

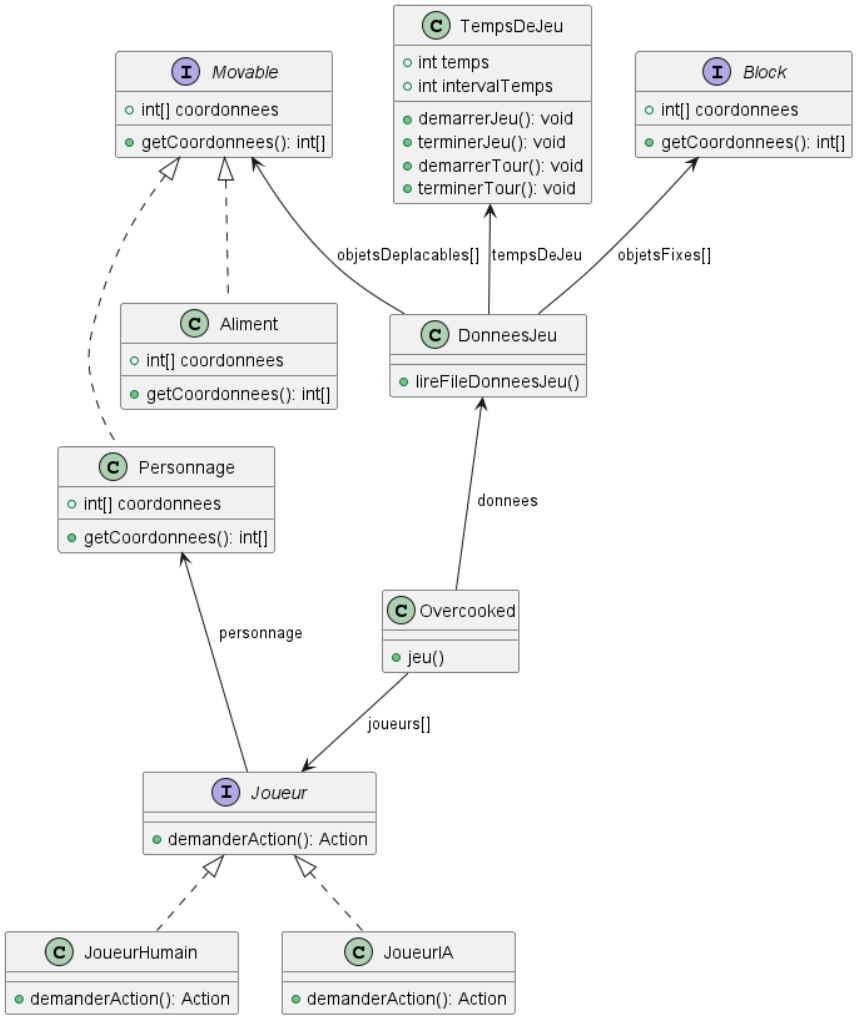
8. Cerveau indépendant (deux IA / IA et vrai joueur)

Disposer d'un cerveau indépendant reprend la même idée que précédemment, mais un seul cerveau qui contrôle une IA. Il faut donc prendre en compte et analyser les actions du deuxième joueur et son état dans le calcul.
Une IA indépendante pourra donc être en coordination avec une autre IA, mais aussi avec un autre joueur. Pour cette partie, nous allons utiliser l'architecture BDI (décrite plus tard)

Fonctionnalités prévues dans le jeu

1. Le joueur peut prendre des aliments et le servir (Service sans assiette),
2. Jouer avec un joueur IA,
3. Le joueur peut couper des aliments,
4. Le joueur peut servir un plat simple (un élément découpé + assiette),
5. Le joueur peut servir des plats composés (plusieurs éléments),
6. Ajout de la cuisson d'un aliment,
7. Le joueur peut servir des plats complexes (plats composés + cuisson),
8. Le jeu va demander des recettes différentes en même temps (plat simple, composé, complexe),
9. Ajout du nettoyage des assiettes,
10. Ajout d'un système qui rend inutilisable un élément lorsqu'il est trop cuit,
11. Le jeu va demander différents plats (sushi, burger) ,
12. Création d'un timer global pour la partie,
13. Ajout d'un timer pour faire le plat,
14. Création d'un système de point,
15. Ajout de la propagation du feu qui rend inutilisable les cases sur lesquelles est le feu,
16. Sélecteur de niveau.

Implémentation de l'application



Code du jeu pour le déroulement de la partie (méthode jeu):

```
Tant que le jeu n'est pas fini :
   - Création d'une liste de threads.
   - Pour chaque joueur dans la liste faire :
      - Création d'un thread associé au joueur, qui effectue les
        actions suivantes :
         - Demander une action au joueur en utilisant l'état
           actuel du jeu.
         - Traiter l'action obtenue.
         - Afficher l'action.
      - Ajout du thread à la liste de thread.
   - Démarrage de tous les threads simultanément.
   - Attendre que tous les threads aient terminé leur exécution
     pour continuer le jeu.
   - Traitement de fin de tour de jeu.

Fin du programme.
```

Le programme utilise une boucle *Tant que* pour vérifier si le jeu est terminé ou non. À chaque itération, des threads sont créés pour chaque joueur, ces threads exécutent des actions spécifiques, à savoir la demande de l'action au joueur.Puis le programme attend que tous les threads aient terminé avant de les exécuter. Ces threads vont donc permettre de synchroniser tous les joueurs pendant un tour et qu'ils puissent choisir leur action du tour en même temps.

Classe TempsDeJeu: Classe utilisée pour gérer tout ce qui concerne le temps, elle se chargera aussi de mesurer le temps du jeu et d'un tour, utile pour mesurer le temps d'exécution de l'IA. Elle sera également chargée de réaliser les affichages des temps.

Fonctionnement de l'IA

L'IA va fonctionner de manière décentralisée, dans l'idéal. Cela signifie qu'il y aura un cerveau par agent. Cela va permettre la gestion avec un joueur réel plus simple. En effet, l'IA n'ayant pas le contrôle de cet agent, il faut qu'elle puisse quand même interagir avec lui. Il faut donc qu'elle puisse choisir ces déplacements en fonction d'un facteur qu'elle ne maîtrise pas. Or avec une IA centralisé, elle va avoir le contrôle des deux agents et par conséquent, elle n'aura pas à gérer ce critère.

Concernant les recettes de cuisine que l'IA va devoir exécuter, elles vont être découpées en objectif et sous objectif (voir Intelligence artificielle - Composition d'un plat)

Le plateau de jeu sera modélisé par un graphe. Ce graphe va prendre en compte 3 critères :

- Le besoin : l'IA va regarder les éléments dont elle a besoin pour réussir la recette et pénaliser fortement les éléments dont elle n'a pas besoin. Par exemple, elle ne va aller chercher un steak pour faire une salade.
- Sa position sur le plateau : L'IA va chercher l'élément le plus proche et non celui qui est à l'autre bout du plateau de jeu
- La position et l'état des autres joueurs : l'IA va devoir prendre en compte si un autre joueur est mieux positionner pour faire une tache. Mais elle va devoir prendre également l'information si celui-ci fait déjà une autre action. Par exemple, l'IA calcule que le plus proche est la tomate (dont elle a besoin). Mais un autre joueur peut être plus proche de la tomate. Dans ce cas, l'IA va plutôt chercher la salade. Mais si le joueur possède déjà quelque chose, alors il ne pourra pas prendre la tomate. Dans ce cas, l'IA va pouvoir aller la chercher.

Algorithmes de recherches

A*

L'algorithme A* été une méthode de recherche utilisée pour trouver le chemin le plus court entre un point de départ et un point d'arrivée dans un réseau ou un graphe. Il commence au nœud de départ, attribue un coût initial, puis estime la distance jusqu'à l'objectif. Le coût total, appelé coût heuristique, est ajouté à une liste de priorité. L'algorithme retire le nœud avec le coût heuristique le plus bas de la liste, l'examine, et s'arrête s'il atteint le nœud d'arrivée. Sinon, il génère de nouveaux nœuds voisins, calcule leurs coûts, et les ajoute à la liste de priorité, à condition que leur coût heuristique soit inférieur à ceux déjà présents. Les nœuds examinés sont ajoutés à une liste fermée. Le processus se répète jusqu'à ce que le chemin optimal soit trouvé.

Les structures de données "liste ouverte" et "liste fermée" peuvent être omises si l'heuristique est à la fois admissible et monotone, ce qui signifie qu'elle ne surestime pas la distance réelle entre les nœuds. Cependant, de tels cas sont rares.
Pseudo code A*

class A* (graphe, début, fin, heuristic)

```
Structure NodeRecord :
Nœud
Connection
coutPlusElevé
coutTotalEstimé

//Initialisation
startRecord = new NodeRecord
startRecord.noeud = début
startRecord.connection = Noeud
startRecord.coutPlusElevé = 0
startRecord.coutTotalEstimé = heuristic estimé(début)

//Initialisation des listes ouvertes et fermée
ouvert = rechercherCheminListe()
ouvert += startRecord
fermée = rechercherCheminListe()

Tant que taille (ouvert) > 0
      actuelle = ouvert.plusPetitElement()
      si actuelle = nœud :: break
      connections = graphe.getConnection(actuelle)
```

```
            Pour chaque connections faire:
                  finNoeud = connection.getToNoeud()
                  coutFinal      =      actuelle.coutPlusElevé      +
connection.getCout()

                  Si fermée.contient(finNoeud)  :
                  finNoeudRecord = fermée.trouver(finNoeud)
                  Continue;

            fermée -= finNoeudRecord
            finNoeudHeuristic  =  finNoeudRecord.couttrotalEstimé  -
finNoeudRecord.coutPlusLoin


                  Sinon si ouvert.contient(FinNoeud) :
                  FinNoeudRecord = open.trouver(FinNoeud)
                  Si FinNoeudRecord.cout  == FinNoeudCout : continue
                  finNoeudEuristic     =     fiNoeudRecord.cout     -
finNoeudRecord.coutPlusLoin
                  Sinon:
                        FinNoeudRecord = new NoeudRecord()
                        FinNoeudRecord.Noeud = FinNoeud
                        finNoeudHeuristic                              =
heuristic.estimate(finNoeud)

                  finNoeudRecord.cout = finNoeudRecord
                  finNoeudRecord.connection = connection
                  finNoeudRecord.coutTotalEstimé  =  finCoutNoeud  +
finNoeudHeuristic

                  Si non ouvert.contient(finNoeud)
                        ouvert += finNoeudRecord
            ouvert -= actuelle
            fermée += actuelle

      Si actuelle.noeud != objectif
            Retourne Null
      Sinon :
            Chemin = []
            Tant que actuelle.noeud != début :
                  chemin += actuelle.connexion
                  actuelle = actuelle.connexion.getFromNoeud()

            retourner inverser(chemin)
```

Structure BDI (Belief, Desir, Intention)

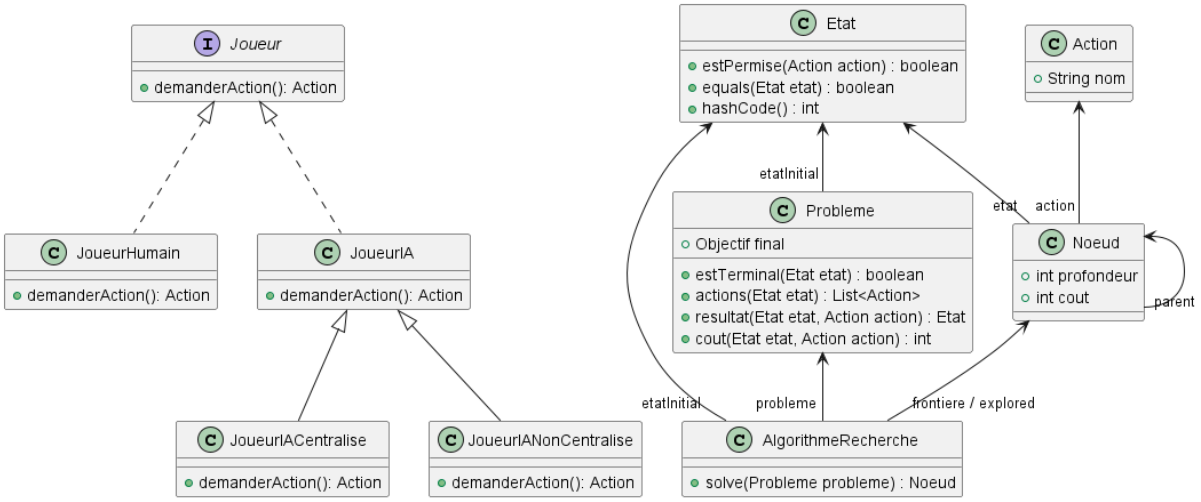
Les croyances d'un agent sont les informations que l'agent possède sur l'environnement et sur d'autres agents qui existent dans le même environnement. Les croyances peuvent être incorrectes, incomplètes ou incertaines et, à cause de cela, elles sont différentes des connaissances de l'agent, qui sont des informations toujours vraies. Les croyances peuvent changer au fur et à mesure que l'agent, par sa capacité de perception ou par l'interaction avec d'autres agents, recueille plus d'informations.

Les désirs d'un agent représentent les états de l'environnement, et parfois de lui-même, que l'agent aimerait voire réaliser. Un agent peut avoir des désirs contradictoires ; dans ce cas, il doit choisir parmi ses désirs un sous-ensemble qui soit consistant. Ce sous-ensemble consistant de ses désirs est occasionnellement identifié avec les buts de l'agent.

Les intentions d'un agent sont les désirs que l'agent a décidé d'accomplir ou les actions qu'il a décidé de faire pour accomplir ses désirs. Même si tous les désirs d'un agent sont consistants, l'agent peut ne pas être capable d'accomplir tous ses désirs à la fois.
Ici, la croyance va être le plateau de jeu (position de chaque élément, élément déjà présent…). Il va également prendre en compte la position des autres joueurs ainsi que leur état (est-ce qu'il possède déjà quelque chose). Le désir va être l'ensemble des recettes. Enfin, l'intention va être le déplacement ainsi que l'objectif but pour réussir la recette.

La structure BDI va être utilisé dans une deuxième partie du projet. La croyance va être le plateau de jeu. Le désir sera l'objectif à atteindre (Par exemple faire un burger. Enfin, L'intention sera le sous objectif à atteindre pour réaliser l'objectif global (par exemple viande cuite).

Implémentation de l'IA envisagée



Cette structure sera amenée à évoluer par la suite puisqu'il nous faudra probablement implémenter différents algorithmes de recherche (qui correspond a A*) ainsi que différents problèmes, en fonction de si l'IA est centralisée ou non. La classe problème est une représentation du problème à résoudre.

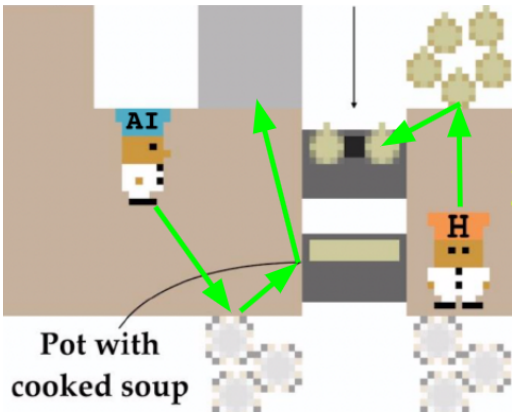
Demander Action:

```
1. Récupère objectif principal
2. Récupère les sous objectif de l'objectif
3. Génère les informations lié à chaques objectif
4. Pour chaque objectif
      a. Créer une classe problème à partir des informations
         (notamment objectif final)
      b. Créer une classe algorithme à partir des informations
         et du problème
      c. Appeler solve de la classe algorithme
      d. Récupérer le coût et le chemin
5. Prendre le coût le plus faible et réaliser son chemin
```

Risques / Difficultés que présente le projet

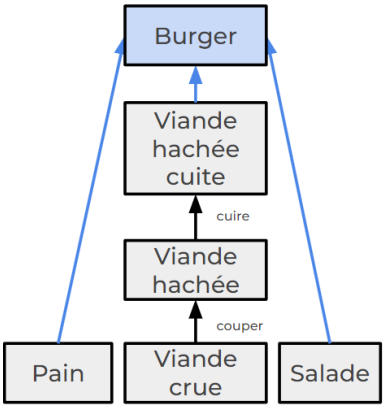
Intelligence Artificielle :

- **Pourquoi je vais de A à B ?**
  - Faut-il tout recalculer à chaque fois ?
  - Tâches en Parallèle ou en ordre
    - Soit ça l'est et un joueur doit cuire un steak avant de prendre le pain
    - Soit ça ne l'est pas et cela n'entraîne pas d'ordre
      - Celle-ci est la meilleure, plus flexible, et plusieurs façons de réaliser un plat



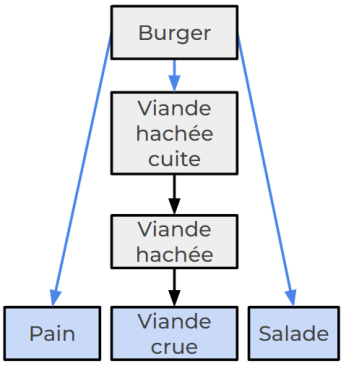
Composition des plats :

- **Des aliments et des transformations**
  - Recette pertinente : Burger
  - Sous recette : Tomate + coupée = tomate coupée
    - On associe un aliment à une transformation qui donne un résultat
  - Plusieurs recettes pour arriver au même plat
  - Comment représenter les recettes ?
    - Une recette n'est pas une suite séquentielle d'étapes à suivre, mais bien un ensemble d'ingrédients nécessaires à la réalisation de la recette. Ces ingrédients peuvent nécessiter des transformations pour arriver au résultat final
  - Comment construire cette séquence ?
  - Pour les recettes, système d'ensemble et sous ensemble, descendre jusqu'aux aliments (si burger, descente vers steak cuit, puis viande coupée, puis viande)



Intelligence artificielle - Composition d'un plat :

- Principe de **Recettes/Sous-recettes**
  - Descente dans la recette finale jusqu'à arriver aux aliments primaires
  - Remontée jusqu'à la recette principale
    - Trouver actions pour remonter (cuisson de la viande, hachage de la viande, …)
    - Fusion des sous-recettes jusqu'à la recette

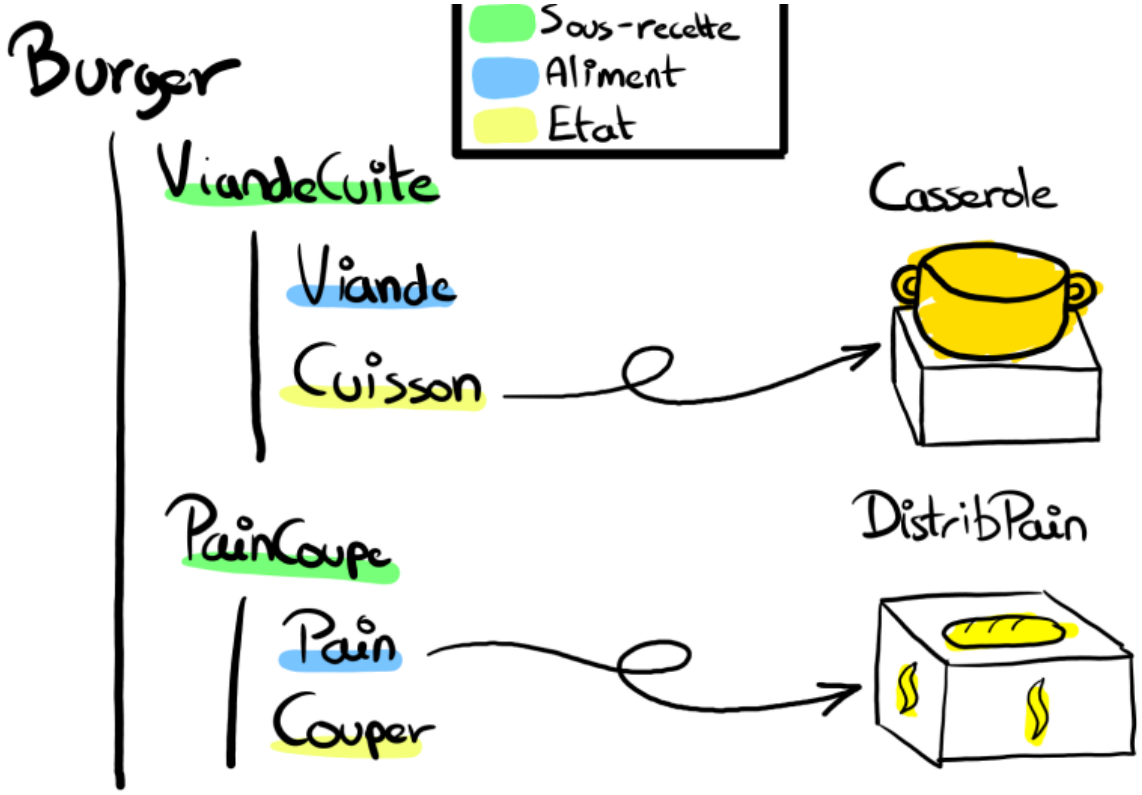


Stockage des données :

- **Aspect IA**
  - Table/Liste : Lecture séquentielle du tableau et interprétation
    - Idéal pour carte, la liste des aliments, les positions des joueurs
    - Recueillement des données plus pratique
- **Aspect pratique**
  - Liste : Lecture plus aisée et séquentielle lors du traitement dans un algorithme
    - Idéal pour la carte et la liste des aliments

Réflexions :

Composition d'un plat :



Nous voyons que pour un plat, plusieurs ingrédients sont nécessaires. Ces ingrédients peuvent être soit des sous-recettes, soit des aliments, soit des états. Par exemple, pour la viande cuite, la recette requiert un aliment (viande) avec un état (cuisson). Ces états et aliments peuvent être trouvés depuis des cases qui seraient liés à ces états/aliments.

Limites de l'IA :

L'IA est dépendante de différents facteurs, qu'ils soient statiques comme les objets ou dynamiques comme les personnages.

L'IA doit calculer le chemin à effectuer en fonction de cela, donc la manière de la limiter et de l'empêcher d'aller à son objectif. Nous pouvons donc avoir différents points de réflexion avec des solutions possibles.

Réflexion :
Si un objectif de réalisation de burger est mis en place, mais qu'aucun pain n'est présent sur la carte (point fixe ou objet), l'IA ne pourrait pas arriver à la fin de son objectif.

Solution :
L'agent pourrait lancer une Exception par exemple. Cela permettrait en plus de comprendre le comportement de l'IA de se rendre compte qu'il manque des objets essentiels à la réalisation d'une tâche.

Réflexion :
Si un collaborateur est humain ou une IA dont l'agent n'a pas le contrôle, elle va prédire son mouvement. Si le mouvement final n'est pas celui qui était prévu, l'agent va devoir réévaluer son calcul, et possiblement changer de direction. À terme, si la réévaluation de direction est systématique, l'agent tournera en rond et n'effectuera rien.

Solution :
Il pourrait être possible, si la réévaluation est trop élevée en fonction du nombre de tours effectués (par exemple, 5 réévaluations en 7 tours), l'agent pourrait passer dans un état de prédiction qui oublie la prévision des mouvements du personnage. Il se charge donc seuls des tâches durant une période donnée (10 tours par exemple).

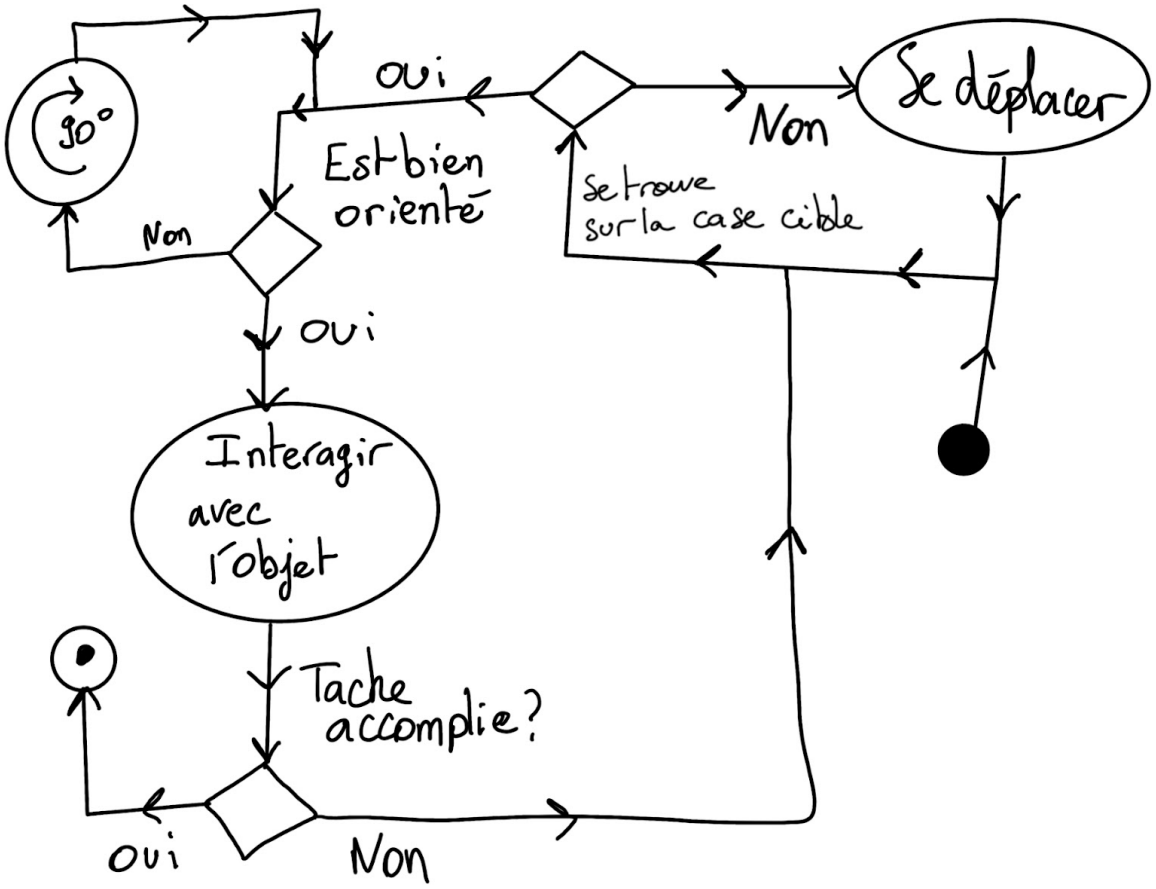
Blocage, si actions effectuées contraires à ce que l'IA pense
Collaboration moindre (on ne bouge pas)
On se met à contre-courant de lui (on lui vole sa nourriture, ses plats)

Objectif de l'itération :

- Cahier des charges → définition détaillée :
    - Du besoin
    - Des fonctions attendues
      → sous forme littérale
- Démarrage de la modélisation des besoins en utilisant les outilsvus en analyse
- Identification des risques et solutions proposées pour les limiter / contourner
- Priorisation des fonctionnalités par itérations

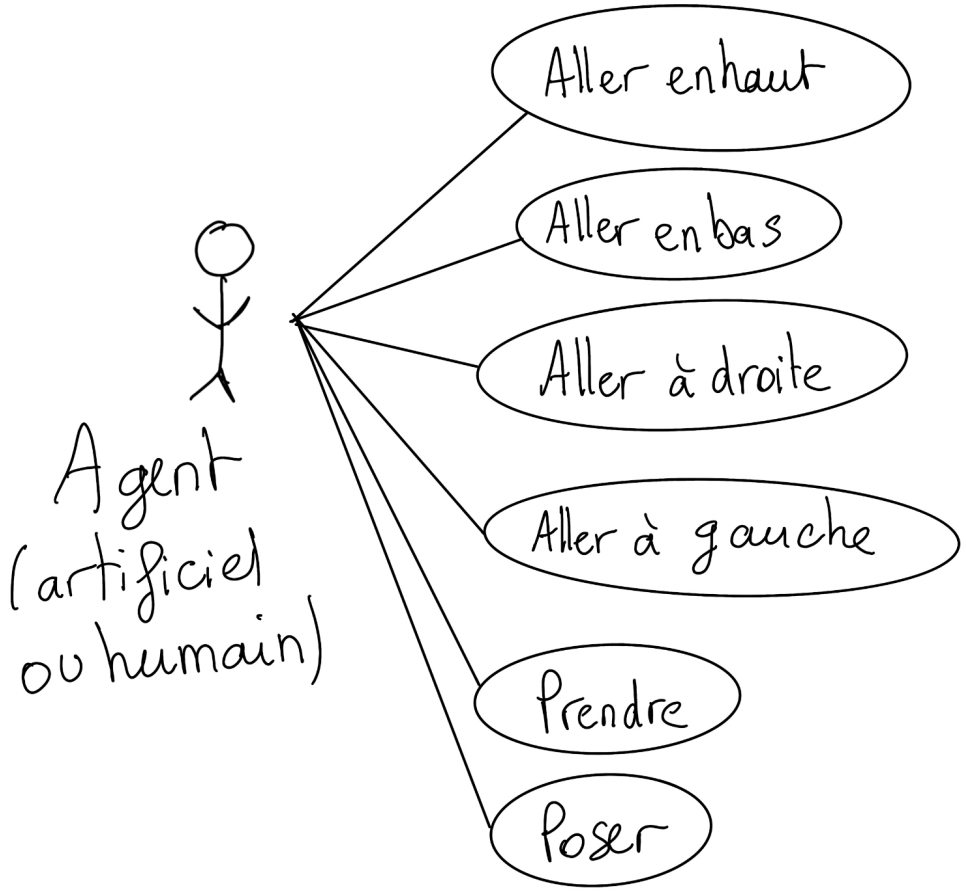
Diagrammes de simulation

Diagramme d'activité des actions à réaliser pour complèter une tâche :



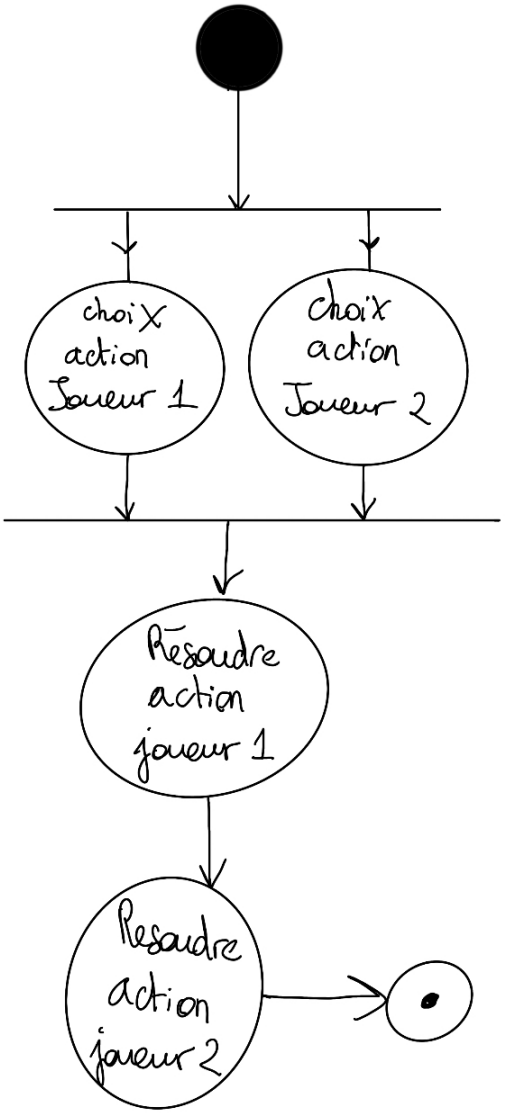
Lorsque l'interaction commence, l'agent se demande s'il se trouve sur la bonne case pour continuer l'interaction, tant que ce n'est pas le cas, il répète l'action. Ensuite, l'opération est exactement la même, mais pour l'orientation cette fois çi. Enfin, lorsque toutes les conditions sont réunies, l'agent peut interagir avec l'objet en face de lui, que ce soit un dépôt de nourriture, une marmite, etc. Si la tâche qu'il avait à faire est terminée, alors l'interaction prend fin, sinon on retourne au point de départ de l'interaction.

Diagramme de cas d'utilisation des actions possibles



Les interactions des joueurs sont assez limitées et se résument au déplacement dans les 4 directions, et à l'interaction (dans les 2 sens) avec un objet.

Diagramme d'activité de la sélection de l'action à réaliser pour le tour suivant

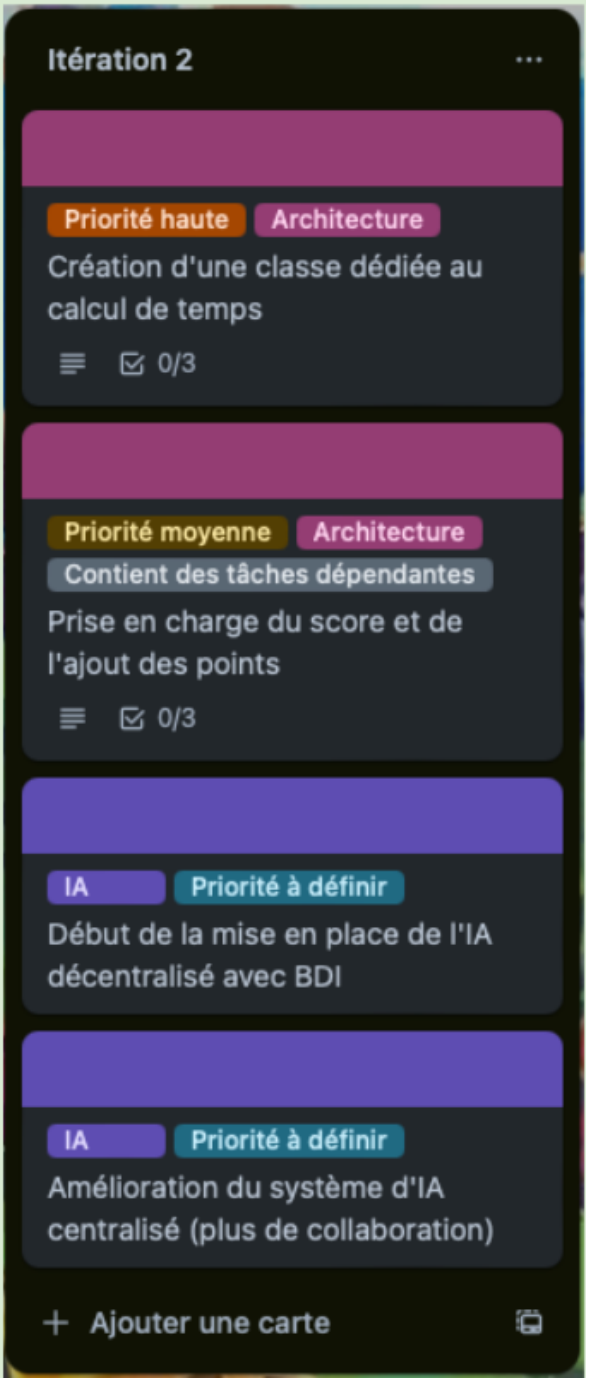


La sélection de l'action à effectuer se fait de manière synchronisée entre tous les joueurs de la partie. Néanmoins, une fois ce choix fait, la résolution des actions se fait dans l'ordre des joueurs. Cette façon de procéder a été choisie pour permettre aux agents artificiels de réaliser leurs calculs de probabilité durant la phase de choix de l'action à faire sans prendre en compte ce que les potentiels joueurs humains puissent avoir choisi.

Planification du projet

Suite de l’architecture

Mise en place concrète de la collaboration



Tâches liées entre elles