SMA Avancé

BDI Agent Implementation

Compte rendu

Clément Potin, Romain Fournier, Yann Marin
Master 2 Informatique, IMAGINE
Université de Montpellier
2021 - 2022

SOMMAIRE

1. Introduction

Pour ce projet nous avons décidé d'implémenter un agent BDI pouvant évoluer dans un environnement qui prendra la forme d'une grille 2D. Le but de ce projet à été pour nous d'implémenter un environnement, ainsi qu'une architecture d'agent BDI dont le code devait être assez générique pour ensuite pouvoir créer un agent capable d'effectuer une livraison.

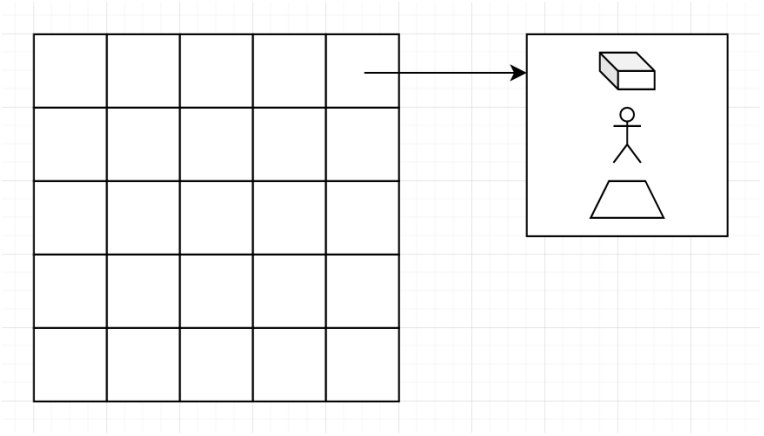
Dans ce compte-rendu, nous aborderons les différents systèmes implémentés à savoir:

- La structure de l'environnement ;
- La structure des croyances, intentions et désires ;
- Le solveur qui, pour un set de croyance et de désire, renvoie une liste d'intentions ;
- Les outils de création de niveaux ;
- La représentation graphique de l'environnement et des croyances de l'agent.

2. L'environnement

Avant de parler de l'agent, nous devons définir l'environnement dans lequel il évoluera. Ce dernier définit la portée de l'agent, les actions qui seront possibles, le format des perceptions, le format des croyances, etc…

Notre environnement est défini comme une grille 2D, où chaque case contient une liste d'entités. Ainsi, on pourra avoir sur une même case un sol, l'agent et un paquet.



Une grille de liste d'entité

Les entités composant le monde sont définies comme des objets ayant un nom, une coordonnée et un attribut "Solid" qui définit si l'entité doit agir comme un obstacle pour l'agent.

Ainsi, pour un paquet on aurait :

- Nom : Paquet
- Coordonnées : X, Y
- Solid : **Non**

Et pour un mur :

- Nom : Mur
- Coordonnées : X, Y
- Solid : **Oui**

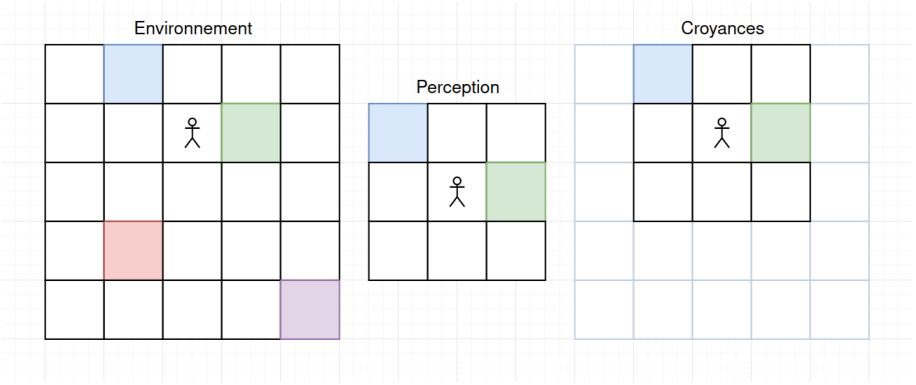
3. Croyances, Intentions et Désirs

A. Les Croyances et Perceptions

Les croyances sont étroitement liées à la représentation de l'environnement. En effet, si on prend un agent BDI devant résoudre des problèmes du type "Tours de Hanoï", il n'aura pas besoin d'avoir une représentation spatiale de son environnement.

Dans notre cas, notre agent aura besoin d'effectuer des opérations dans l'espace de son environnement, il devra se déplacer, ouvrir des portes, porter des boites… Il est important dans notre implémentation de stocker et d'ordonner ces informations d'une manière qui sera utile à notre agent. Ainsi, étant donné que la position et l'état des entités peuplant l'environnement est primordiale au fonctionnement de notre agent, le format choisi pour représenter les croyances est le même que l'environnement, à savoir une grille d'entités.

Les perceptions deviennent alors triviales. Il suffit de copier l'état de l'environnement autour de notre agent et de le comparer à nos croyances. On peut ainsi détecter de nouvelles entités, détecter les changements ainsi que les entités manquantes. Nous gardons donc pour croyance un état du monde tel que l'agent se le représente.



Un agent qui perçoit le monde

B. ***Les Intentions***

Les intentions de notre agent sont représentées comme une liste d'actions. La première intuition qu'on pourrait avoir serait de définir un set d'actions pour l'agent. Il pourrait marcher à gauche, à droite, en haut, en bas, il pourrait ouvrir une porte, porter un colis… L'avantage de cette approche est que notre agent est bien défini. On connaît parfaitement sa portée et l'implémentation semble simple.

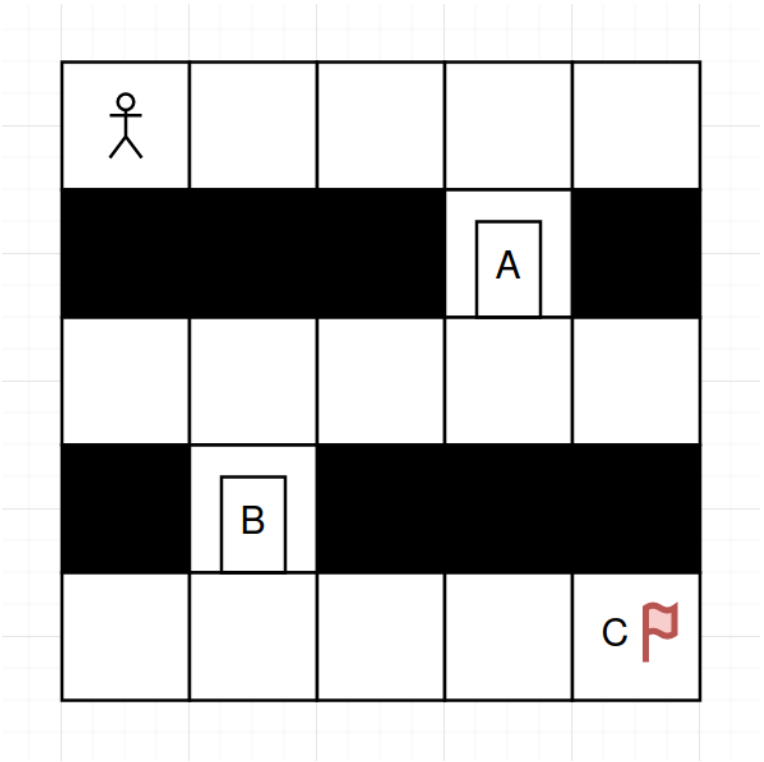
Cependant, une telle représentation réduirait les intentions de notre agent à des actions atomiques. Si on prend l'exemple ci-dessous, un agent ayant pour mission d'aller regarder le drapeau (C).

Là où on voudrait que notre agent puisse avoir les intentions suivantes :

- Ouvrir la porte (A), Ouvrir la porte (B), Regarder le drapeau (C)

Il aurait à la place:

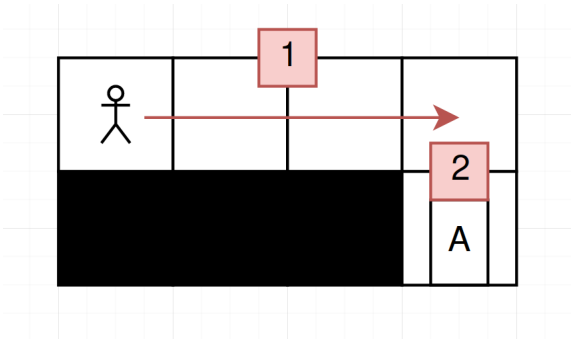
- Droite, Droite, Droite, Ouvrir, Bas, Bas, Gauche, Gauche, Ouvrir, Bas, Bas, Droite, Droite, Regarder



Le périple d'un agent

Nous avons donc choisi une approche différente. Les actions sont définies dans les entités. Une porte peut être ouverte et fermée, un colis peut être ramassé et posé, un drapeau peut être regardé…

De plus, si on prend l'exemple ci-dessous, l'intention "Ouvrir la porte (A)" cache en réalité deux actions distinctes. La première est "Aller devant la porte (A)", la seconde est l'action que nous souhaitions, "Ouvrir la porte (A)". Ainsi, dans notre implémentation, une intention se réalise en deux temps, dans un premier temps (1), nous réalisons un pathfinding vers l'entité cible, puis (2) l'action est effectuée.



Exécution d'une intention

À noter que nos intentions sont calculées depuis l'espace des croyances, si l'agent n'as pas conscience d'un mur sur sa route, cela ne l'empêchera pas de tracer un chemin vers une entité. C'est seulement lorsqu'une perception viendra invalider le chemin qu'il à tracé qu'il réévaluera un nouveau chemin, voire réévaluera ses désirs et intentions.

Cette représentation des intentions nous permet de simplifier les actions sous une forme compréhensible pour un humain. Un agent ayant pour intention "Ouvrir la porte (A)" peut le dire à un humain de façon compréhensible pour ce dernier, tout le contraire d'actions atomiques comme "Droite, Droite, Droite, Ouvrir".

De plus, cela retire des actions inutiles de notre espace de recherche. Si l'on cherche la suite d'action optimale dans un arbre, le format d'intentions que nous avons choisi permet d'aller beaucoup plus vite que sur un set d'actions atomiques.

En effet, si on prend l'exemple cité plus haut (*Le périple d'un agent*), dans le cas où il faudrait à l'agent 3 étapes au moins pour réaliser son désir et où notre agent peut réaliser 3 actions à tout moment, notre arbre de recherche est très réduit, nous avons $3^3 = 27$ états du monde à tester.

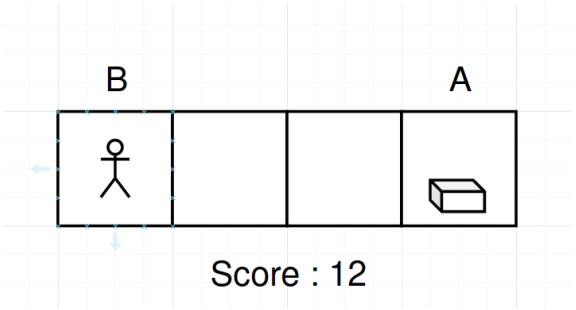
Si on prends un set d'actions atomiques (Droite, Gauche, Haut, Bas, Ouvrir, Regarder), et qu'il faut 14 étapes au minimum pour arriver à la solution minimale, on aurait $6^{14} = 78364164096$ états du monde à tester.

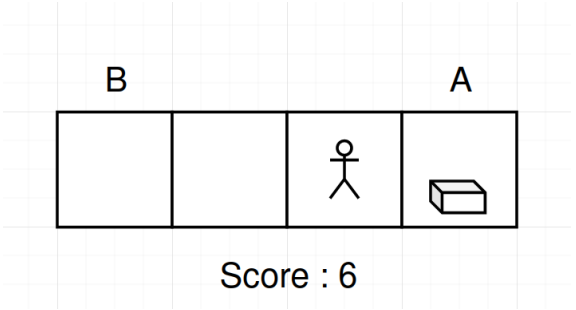
Enfin, cette implémentation rend l'ajout d'intentions générique, il suffit lors de la création d'une entité de définir les actions possibles sur cette entité, ainsi que les comportements pour chaque action.

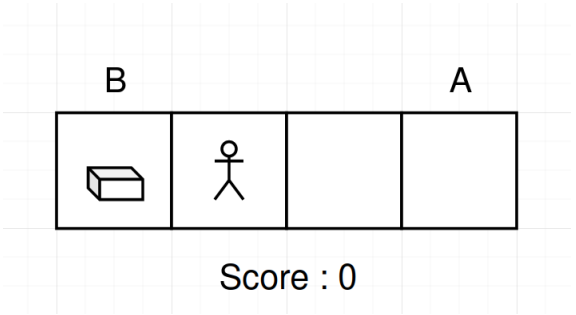
C. ***Les Désirs***

Enfin, nos désirs sont représentés par des fonctions d'évaluation. C'est certainement le point le plus trivial de cette partie. Pour un état du monde et un agent, un désir est capable de donner une priorité (est-ce que ce désir est urgent ou pas) et un score (est ce que cet état du monde satisfait ce désir). Nous avons choisi un format où il faut minimiser un score.

Ainsi, si on prends l'exemple ci dessous, on pourrait associer au désir "Livrer la boite (A) au point (B)" les scores suivants :





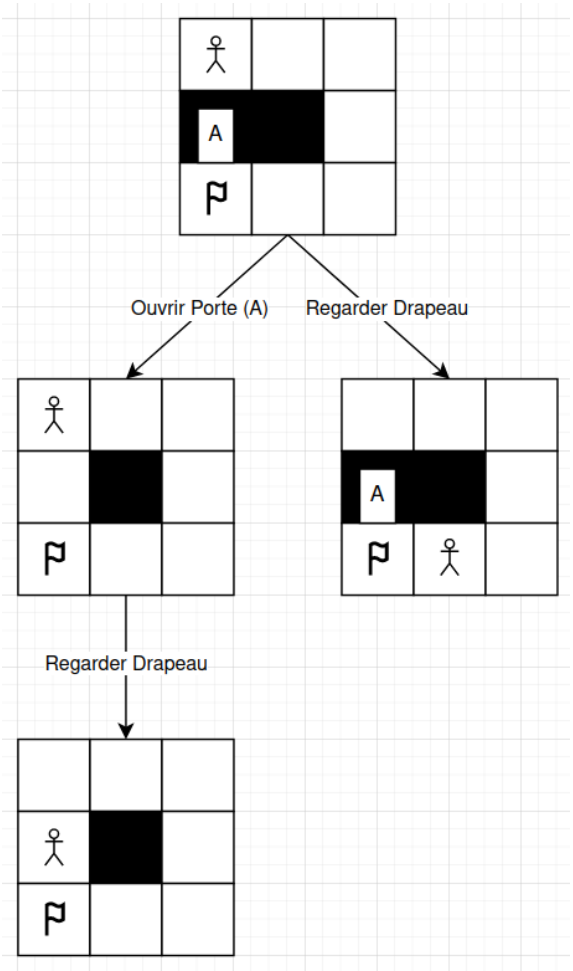


Ce format nous permet de rendre générique les désirs. En effet, il nous suffira de définir deux fonctions pour définir un nouveau désir. On peut ainsi imaginer le désir "Fuir la boîte maléfique (B)", qui retourne comme score l'inverse de la distance à la boîte !

4. Le Solveur

Pour le solveur, nous avons choisi de traiter notre espace de recherche comme un arbre pour pouvoir y appliquer des algorithmes de recherche classique. À chaque action réalisée, nous nous trouvons dans un état enfant de l'état précédent.

Si on prends un exemple simple, nous pouvons représenter l'espace de recherche ainsi :



Exemple d'espace de recherche

Pour des cas simples, une recherche en largeur suffirait à trouver toutes les solutions. Il faudrait ensuite trier les solutions par coût, un exemple de fonction de coût serait la somme des actions additionnée à la somme des déplacements, si on marche beaucoup ou qu'on fait beaucoup d'actions, on serait désavantagé par rapport à une solution plus "simple".

Cependant, nous avons choisi d'utiliser un recuit simulé. L'idée derrière l'utilisation d'un algorithme de méta-heuristique était de pouvoir garder des temps d'exécutions en temps interactif même pour les cas les plus complexes. Le point négatif, en revanche, est que l'agent peut trouver une solution qui est loin d'être optimale.

5. ***L'agent***

Nous avons donc un agent évoluant dans un environnement 2D, et qui doit répondre à la notion de pathfinding afin de réaliser les intentions en deux étapes vues plus haut. Notre boucle est donc légèrement différente de la boucle d'un agent BDI générique.

En effet, à chaque "tick" (mise à jour du monde et donc des agents qui y évoluent), l'agent pourra non seulement percevoir son environnement, évaluer ses intentions et désirs, mais aussi marcher s'il peut se déplacer et, à défaut, effectuer une action sur l'environnement. De plus, si une perception vient "couper" un chemin ou ajoute des informations importantes, l'agent devra réévaluer ses désirs et intentions. En pseudo code, nous avons donc :

```
Percept ← Perception(X, Y, 1)
Δbeliefs ← AddPercept(Beliefs, Percept)
ici, Δbeliefs est le nombre de changements dans les croyances. certaines entitées comptent pour plus

Check if the path is empty or obstructed
if (Path is empty, or Path is obstructed)
        generateDesire ← True

Process Desires if needed
if (no Desire, or generateDesire, or Δbeliefs > treshold)
        Desire ← ChooseDesire(Beliefs)
        generateDesire ← False
        generateIntentions ← True

Process Intentions if needed
if (no Intentions, or generateIntentions, or Δbeliefs > treshold)
        Intentions ← MakePlans(Beliefs, Desire)
        generateIntentions ← False
        Path.Clear()

Generate a path if needed
if (Path is empty, and there are Intentions)
        Path ← PathFind(Beliefs, Intentions[0])

Walk if we can
if (Path is not empty)
        MoveTo(Path[0])
        Path.Remove(0)
If not, do an action if we can
else if (Intentions is not empty)
        Do(Intentions[0])
        Intentions.Remove(0)
```

6. Représentation visuelle de l'environnement

Pour ce projet, une attention particulière a été portée sur le rendu visuel. En effet, nous souhaitions que l'environnement, l'agent, ses actions et connaissances soient représentés d'une façon claire et intuitive. C'est pour cette raison que nous avons choisi le moteur de jeu "*Godot*" pour représenter notre simulation.

L'utilisation d'un moteur de jeu nous permet un rendu visuel bien plus compréhensible qu'un simple affichage console, grâce aux nombreuses fonctionnalités qu'il propose.

Notre choix s'est porté sur *Godot* plutôt qu'*Unity* ou *Unreal Engine* pour son aisance d'utilisation ainsi que la légèreté du moteur.

A. La carte

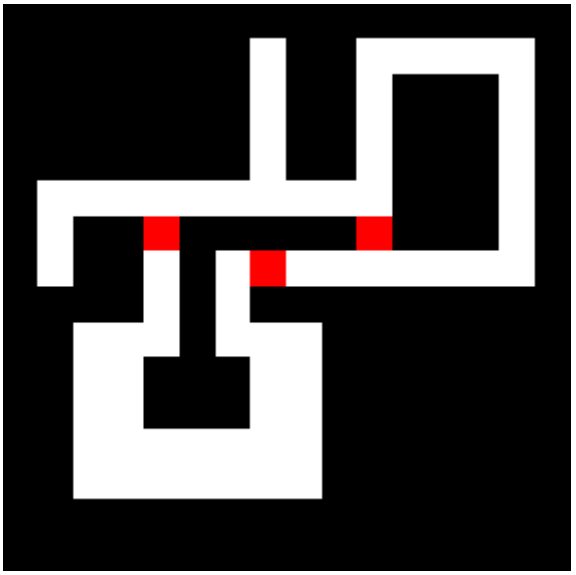
a. Bitmaps

Pour créer nos cartes, nous avons choisi d'implémenter un système de bitmaps. Le principe de ce genre de systèmes est de représenter nos cartes par des images dont chaque pixel correspond à un bloc (une "tile") de notre "tilemap". La couleur (ou plus précisément, le code couleur) du pixel définit alors quel objet/entité placer sur la carte : mur, sol, …

Dans notre cas, nous avons utilisé les associations suivantes :

	Couleur	**Code couleur**
Mur	Noir	#000000
Sol	Blanc	#FFFFFF
Portes	Rouge	#FF0000

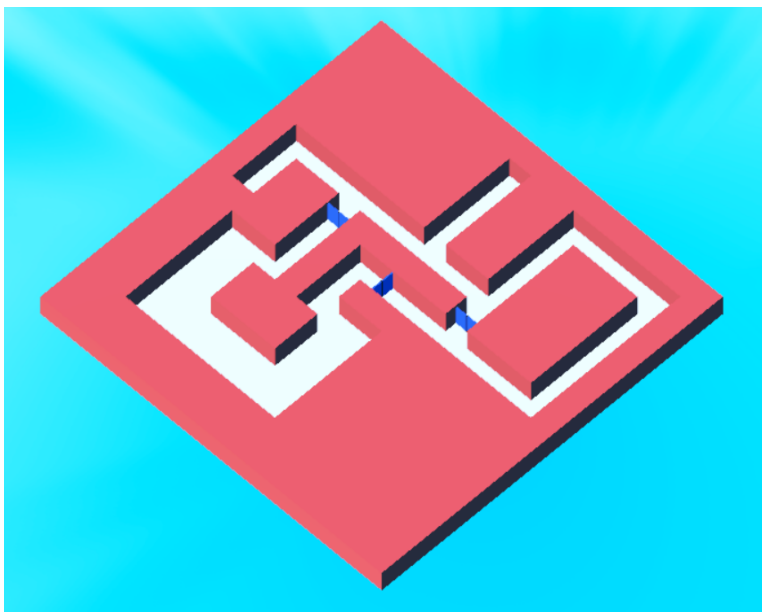
En utilisant ces associations, on peut créer des cartes telles que :



Bitmap

b. Carte “solide”

En utilisant cette bitmap, on peut instancier différents objets pré-créés aux bonnes coordonnées pour représenter la carte. En associant des objets en 3D à une caméra orthographique dont la position s’ajuste par rapport à la taille de la carte, on obtient cette représentation pour la bitmap précédente :



Carte “solide”

c. Carte “fantomatique”

Pour différencier le monde réel (ce qui existe, mais que l’agent n’a pas encore découvert) du monde connu (l’environnement que l’agent a découvert et enregistré), nous avons créé une version “fantomatique” (flottante et partiellement transparente) du monde.

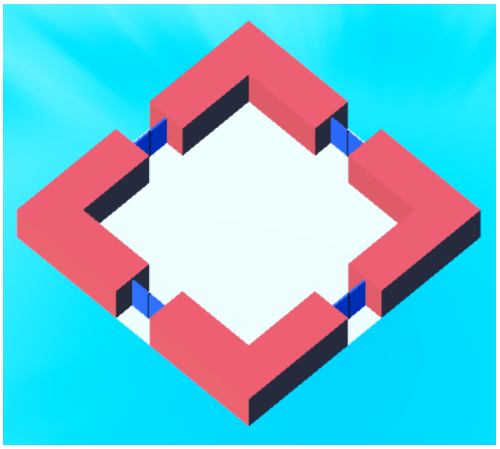
La même carte, représentée de façon fantomatique :

Carte “fantomatique” (gif animé)

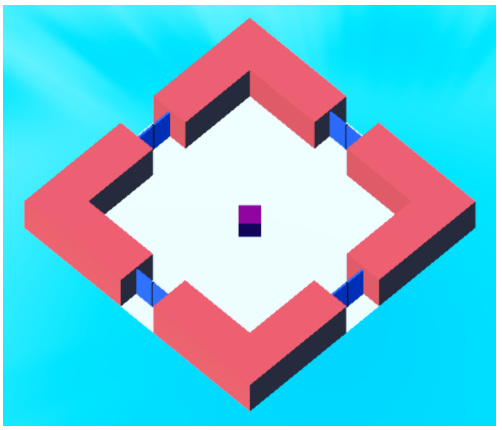
B. Les entités

Les drapeaux mis à part (n’ayant été utilisés que lors du développement et des tests pour ce projet), 6 entités existent dans nos environnements :

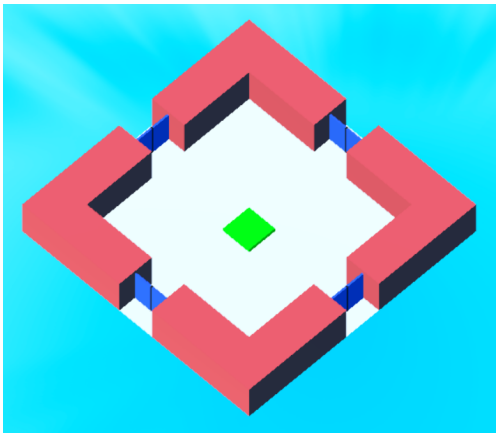
- Les murs, en rouge (l’agent ne peut pas interagir avec) ;
- Le sol, en blanc (l’agent peut marcher librement dessus) ;
- Les portes, en bleu foncé (l’agent peut les ouvrir/fermer, et ne peut passer que si la porte est ouverte) ;
- Les paquets, en violet (l’agent chercher à récupérer les paquets) ;
- Les points de livraison, en vert (l’agent chercher à déposer le paquet qu’il a avec lui sur un point de livraison) ;
- L’agent, en bleu clair.



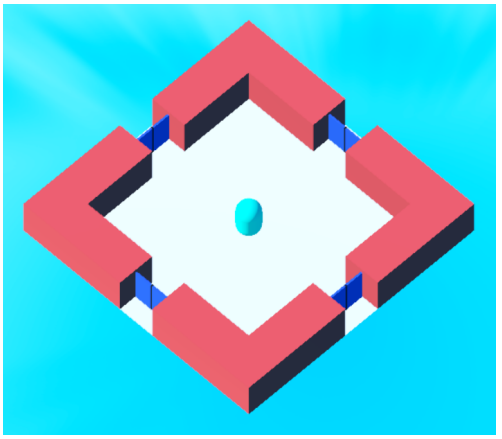
Salle vide



Salle avec paquet



Salle avec point de livraison



Salle avec agent

Il faut noter qu'ici, la carte est simulée comme étant entièrement découverte par l'agent. Si ce n'était pas le cas, voici ce que l'agent verrait (la partie fantôme n'étant pas découverte et donc inconnue de notre agent) :

Vision réelle de l'agent (gif animé)

C. ***Simulations***

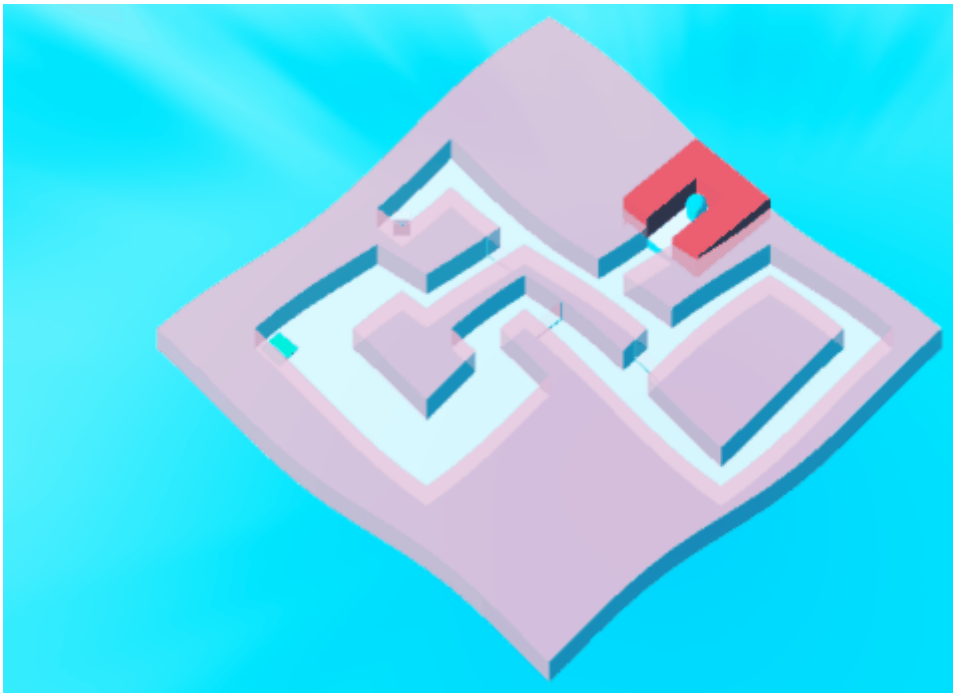
Une fois notre agent BDI implémenté et la représentation visuelle liée à ce dernier, on peut se prêter à différentes simulations, et observer comment l'agent se comporte d'une simulation à l'autre.

Sur une petite carte :

Simulation 1 (gif animé)

Simulation 2 (gif animé)

Sur une carte plus élaborée :



Simulation sur la carte d'origine (gif animé)

7. ***Liens utiles***

Répertoire GitHub :
https://github.com/Romimap/BDI-Agent-Implementation

Lien du Google Docs avec version animée des gifs :
https://docs.google.com/document/d/16g3OLbcFvrNeKGpm1DrwtQJs0egMhgM7X7Fsnl2cqD8/edit?usp=sharing