Rapport de Projet

Explications qualitatives de trajectoires de robots

Dans le cadre du
MASTER de Sorbonne Université

Spécialité
Informatique

Parcours
ANDROIDE

ZUO Nicolas	Étudiant
YANG Zitong	Étudiante
FOLTYN Axel	Étudiant
BLOCH Isabelle	Cliente/Encadrante
BEYNIER Aurélie	Cliente/Encadrante
MAUDET Nicolas	Client/Encadrant

Table des Matières

1 Introduction

Notre introduction reprenant les notions importantes du cahier des charges, nous avons mis celui-ci en annexe et non au début du rapport.

L'intérêt pour la robotique croît depuis plusieurs années, tout d'abord dans le domaine industriel comme l'automobile ou l'aviation puis maintenant dans le domaine domestique avec la commercialisation des drones, et celles des robots comme les aspirateurs autonomes et bientôt les voitures autonomes. Ces robots ont des tâches variées à effectuer et l'interaction avec l'humain est fondamentale. Ces échanges entre le robot et l'humain, sont parfois difficiles à comprendre du point vu de l'Homme, ce qui nous amène à des utilisateurs non experts de ne plus comprendre les choix pris par le robot.

L'explication qualitative de trajectoires de robots est un problème majeur et le sujet même de notre projet, qui consiste comme son nom l'indique d'expliquer qualitativement une trajectoire d'un robot, plus concrètement, notre robot devra expliquer de façon linguistique, le plus naturellement possible pour qu'il soit compréhensible par tout le monde, des choix de trajectoire qu'il a choisis par rapport à d'autres trajectoires possibles. Partant de ce principe, nous avons fait nos recherches dans ce domaine, et nous avons constaté qu'il a très peu de documentation sur ce domaine qui semble pourtant très important et présent dans la majorité des domaines robotique.
Nos recherches concordent avec les recherches de nos Clients/Encadrants. Lors de leur recherche dans ce domaine, ils se sont rendu compte que les articles et recherches parlant de ce sujet sont très limités, où il y a vraiment très peu de recherche dans ce domaine d'explication qualitative d'une trajectoire, par une méthode linguistique.

Nous sommes ici dans le cadre d'un projet de Master 1 d'une durée de 4 mois et notre travail est alors de faire un premier travail sur ce domaine, pour voir les approches utilisées pour répondre à notre sujet.
Dans le cadre de notre projet, nous nous intéressons à l'explication du chemin que le robot emprunte et de l'explication du choix sélectionné, et pour arriver à cela, on devra tout d'abord faire une génération d'un ou plusieurs chemins, mais aussi d'un affichage facilement compréhensible pour afficher les descriptions et explications que le robot fait au cours du trajet.

1.1 Description l'Environnement

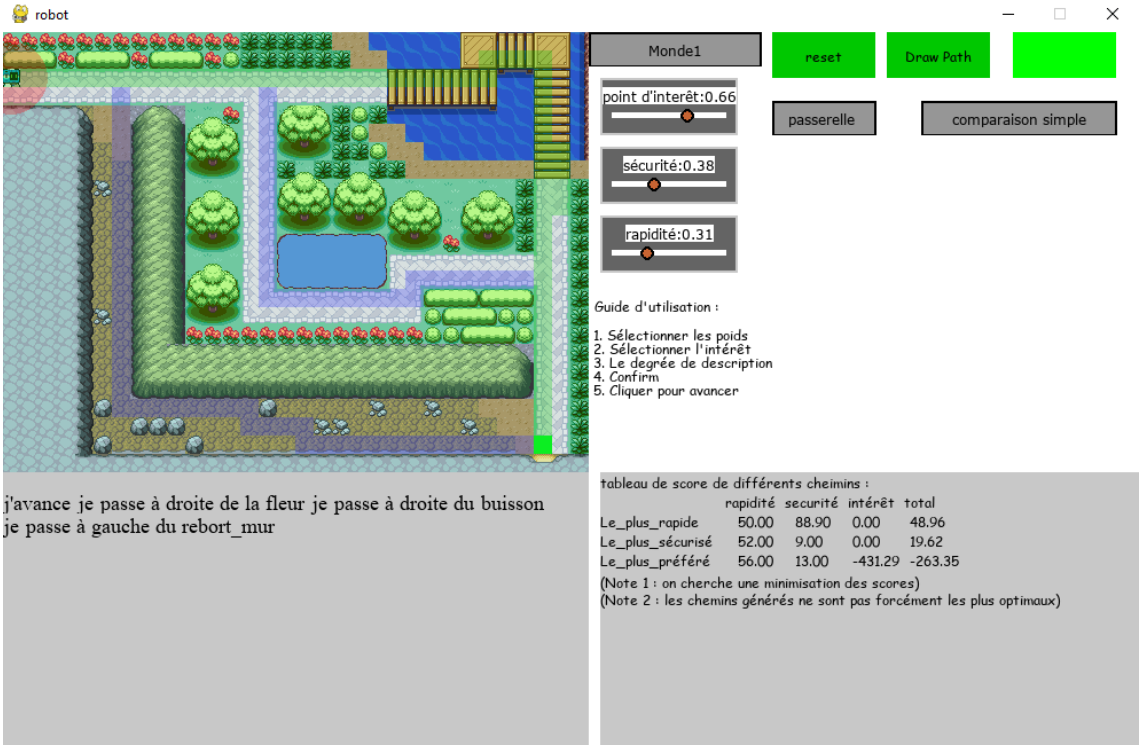


FIGURE 1 – Version final de notre fenêtre d'application

Notre environnement ou notre fenêtre de notre application, comporte plusieurs zones, la partie haut-gauche, sert de représentation de notre environnement/terrain où notre robot va se déplacer à l'intérieur. La partie bas-gauche, sert de zone de dialogue du robot, là où il est en train de décrire de façon linguistique ces actions et de potentielle explication a des moments clés. Et la dernière partie, la droite, sert au paramétrage et aux différents scénarios et de notre robot, avec un tableau des scores des différents chemins générés en bas. Pour finir pour faire avancer notre robot une fois que vous avez appuyé sur "confirm", il suffit de faire des clics gauches, qui feront avancer le robot d'une case a chaque fois.

1.2 Contraintes

Notre environnement est une surface plane et statique, représentée sous forme de grille. Notre robot occupe entièrement une case et a un rayon d'interaction définie au départ. Les objets occupent aussi entièrement une case, mais il peut y avoir plusieurs objets pour représenter un même objet, le robot ne peut aller que sur les cases dites "canPass" définie dans un fichier XML/TSX, par exemple un robot peut marcher sur une case nommée terre qui a cet attribut à "True", mais ne peut pas marcher sur des fleurs, car ils ont l'attribuent à "False". Le robot connaît sa position initiale ainsi que le point d'arrivée.

D'un point vu du niveau de l'interaction souhaité avec l'utilisateur on souhaite, que l'utilisateur puisse donner différents chemins possibles, pour plusieurs comparaisons de trajectoire possible, et aussi un certain de niveau de généricité des outils et fonction généré pour avoir un projet le plus maniable possible sur plusieurs points, comme par exemple changer le style linguistique ou le degré de précision (exemple : "très proche", "un peu moins proche" ...)

2 Différents scénario

2.1 Scénario 1 : Chemin Existant...

Notre tout premier scénario est très basique et qui ne sera pas retenu, car remplacé par un prochain scénario qu'on abordera plus tard, il est constitué d'une carte, carte faite a la main sur le logiciel Tiled, ici notre point de départ, représenter en rouge, le point d'arrivée en vert et les cases bleues représente le chemin tracé à la main, où notre robot devra suivre ce chemin et tout au long du trajet notre robot affichera dans la zone de dialogue, des descriptions simple, "J'avance", "... je passe à gauche des fleurs", "... je tourne à droite", etc. On a fait une version où notre robot explique en une seule fois tout le chemin dès le départ, mais qui ne s'avère pas assez réaliste et on a fait une version où tout au long du trajet notre robot décrit son environnement a chaque itération.

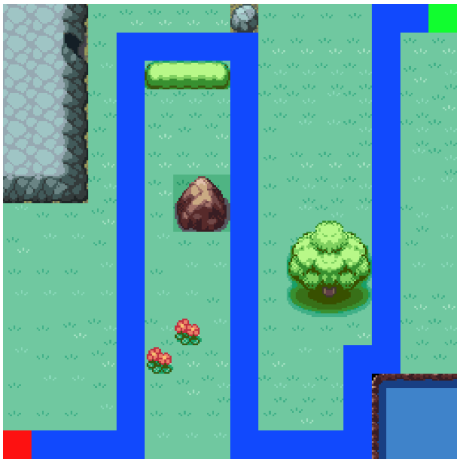


FIGURE 2 – Première carte

2.2 Scénario 2 : ... Vers une multitude de Chemin Existant...

Après avoir réussi à faire des descriptions simples sans réelle explication derrière, nous avons réfléchi à comment faire pour que notre robot puisse expliquer pourquoi il a choisi ce chemin en particulier et non un autre, et ainsi expliqué les spécificités du chemin choisi, on a alors très vite compris qu'il fallait d'autres chemins possibles pour pouvoir avoir des comparaisons possibles avec notre chemin actuel. On est donc parti du même principe que le scénario 1, mais avec plusieurs chemins possibles qui se sépare au cours du chemin. On a alors rajouté la notion d'un "Score de rapidité" qui est le nombre de cases nécessaire pour atteindre l'arrivée, d'un "Score de sécurité" qui représente le niveau de dangerosité du chemin (par exemple un chemin

passant proche des rochers aura un score de sécurité très mauvaise, par rapport à ceux qui passent à côté des fleurs ou des arbres qui ne sont pas dangereux) et d'un dernier paramètre le "Score de points d'intérêt" de l'utilisateur, par exemple l'utilisateur accorde plus d'importance à certain objet sur la carte, alors plus le chemin passe à côté des objets ciblés par l'utilisateur alors plus le score sera meilleur.

Et donc avec tous ces scores, l'utilisateur peut définir au début le poids qu'il met sur ces différents paramètres possibles, un poids élevé en sécurité, fera que notre robot essayera d'éviter au maximum les chemins qui passent proches des objets dangereux. On a ensuite utilisé la méthode du scénario précédente, à chaque itération de notre robot sur le chemin, il affichera une description simple, mais en plus de cela a chaque première intersection où notre meilleur chemin choisi, grâce au meilleur score global de tous les paramètres, se sépare d'un autre chemin il déclenche une explication, où le robot détail pourquoi il a pris ce chemin par rapport aux autres chemins, en comparant les scores de rapidité, sécurité et point d'intérêt.

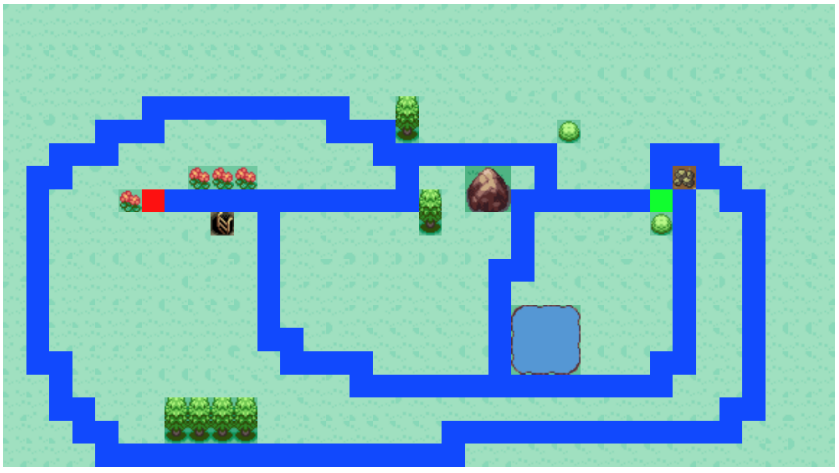


FIGURE 3 – Chemin multiple

2.3 Scénario 3 : ... Vers une Génération Automatique...

On a ensuite voulu créer une génération automatique de plusieurs chemins, au lieu de devoir dessiner à la main sur un autre logiciel, tel que Tiled pour ensuite travaillé dessus, et donc fini les chemins bleus tracer à l'avance !

On a donc généré un plus court chemin où l'on a toujours un point de départ et un point d'arrivée, avec un algorithme **A*** implémenter de A à Z, pour ensuite le manipuler facilement et l'adapter pour nos cas particuliers, comme par exemple l'ajout

de poids sur les cases où il y a du danger, ce qui fait que l'algorithme va essayer d'éviter au maximum ces cases au risque d'avoir des coups élevés en sortie, on a donc utilisé ce même principe où l'on génère un deuxième chemin qui sera le chemin le plus sécurisé et rapide possible.

En résumé nous avons maintenant deux chemins générés automatiquement, mais cela ne suffit pas pour avoir des comparaisons plus variées, on a alors fait une troisième génération qui se base sur le poids de tous les paramètres qu'on donne au départ. ce qui nous donne au total trois chemin généré automatiquement à chaque parcours de chemin, ce qui permet à notre robot de choisir le chemin avec le meilleur score possible (représenté par le chemin vert sur les figures 3 & 4) et ainsi à chaque première intersection où notre robot va se séparer des autres chemins il va déclencher une explication linguistique de pourquoi il a choisi le chemin (vert) par rapport aux autres chemins (bleu), il faut aussi noter pour des raisons de clarté quand on arrive à une intersection, les chemins comparés seront automatiquement sélectionnés et change de couleur pour savoir dans la fenêtre de dialogue, quelle explication correspond à quelle couleur de chemin (figure 4) (on détaillera plus en détail, par la suite de comment marche une explication linguistique qualitative).

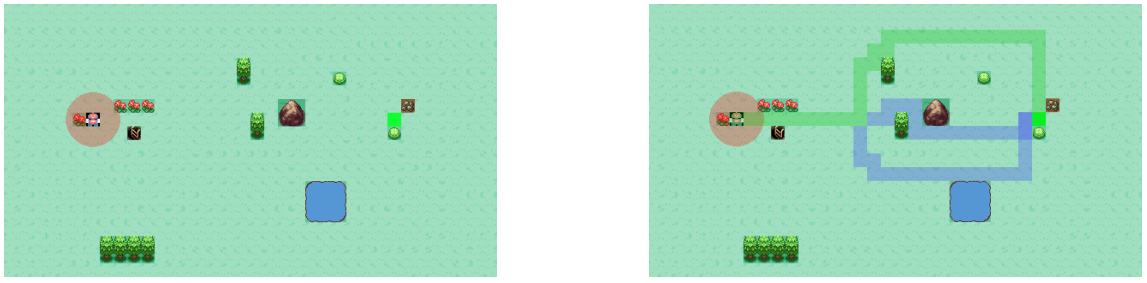


FIGURE 4 – Avant & Après la génération automatique

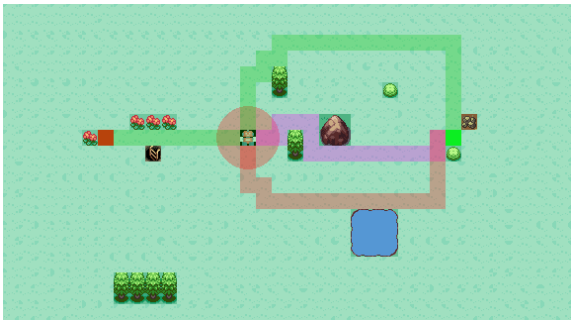


FIGURE 5 – Arriver à une intersection

2.4 Scénario 4 : ... Vers une diversification des chemin...

Avoir trois chemins ne nous a pas suffi et limite notre explication à des chemins prédéfinis en fonction des paramètres définis au départ, ce qui fait que l'utilisateur ne peut pas par exemple testé tous les chemins possibles que lui aurait voulu que le robot parcours, on a alors repris le travail fait au départ du projet, qui est le scénario 1 et 2, où on va reprendre le concept du "Blue Path" ou chemin dessiner à la main, mais cette fois on peut le dessiner directement sur la fenêtre de l'application, pour que ça soit beaucoup plus interactif (figure 5), ce qui rajoute en plus des trois chemins générés automatiquement le ou les chemins dessinés par l'utilisateur, l'algorithme calculera automatiquement tous les chemins possibles depuis le dessin de l'utilisateur, a quelque détail près. Par exemple sur la figure 6, notre algorithme a détecté qu'il avait deux chemins possibles et il choisira celle avec le meilleur score possible des chemins dessinés et quoi qu'il arrive notre algorithme suivra un chemin dessiné par l'utilisateur s'il en a une, et il fera l'explication linguistique qualitative par rapport au chemin dessiné par l'utilisateur et tous les autres chemins dessinés par l'utilisateur et les chemins non sélectionnés qu'ont été dessinés par l'utilisateur (représenté par un chemin vert foncé sur la figure 6).

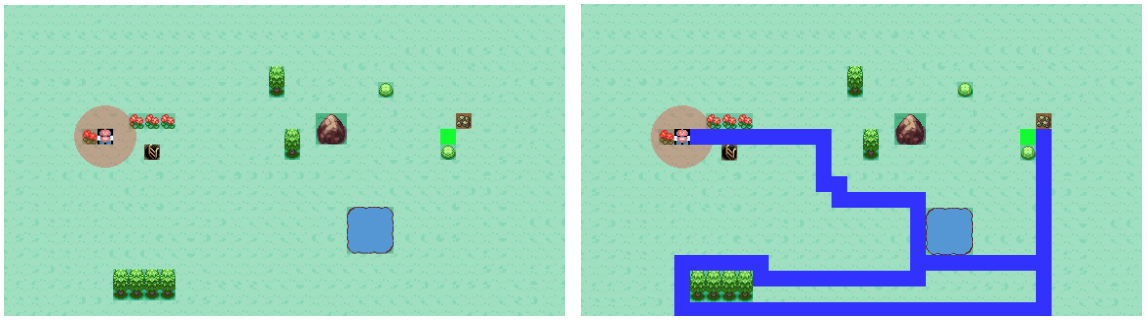


FIGURE 6 – Avant & Après le dessin par l'utilisateur

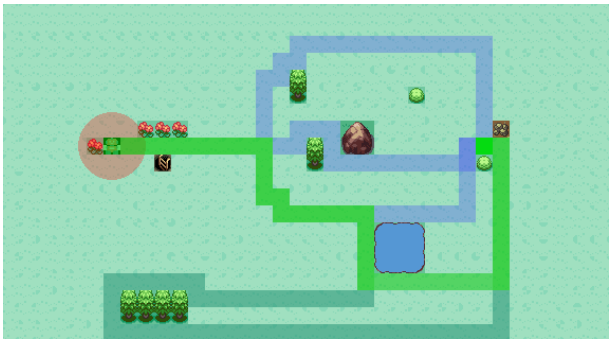


FIGURE 7 – Après confirmation du fin du mode dessin

2.5 Scénario 5 : ... Vers des optionalités d'explication Linguistique.

Nous avons par la suite réfléchi à comment implémenter un programme pouvant être le plus adaptable possible et pouvant choisir un degré d'explication, ce qui nous ramène à notre dernière version finale.

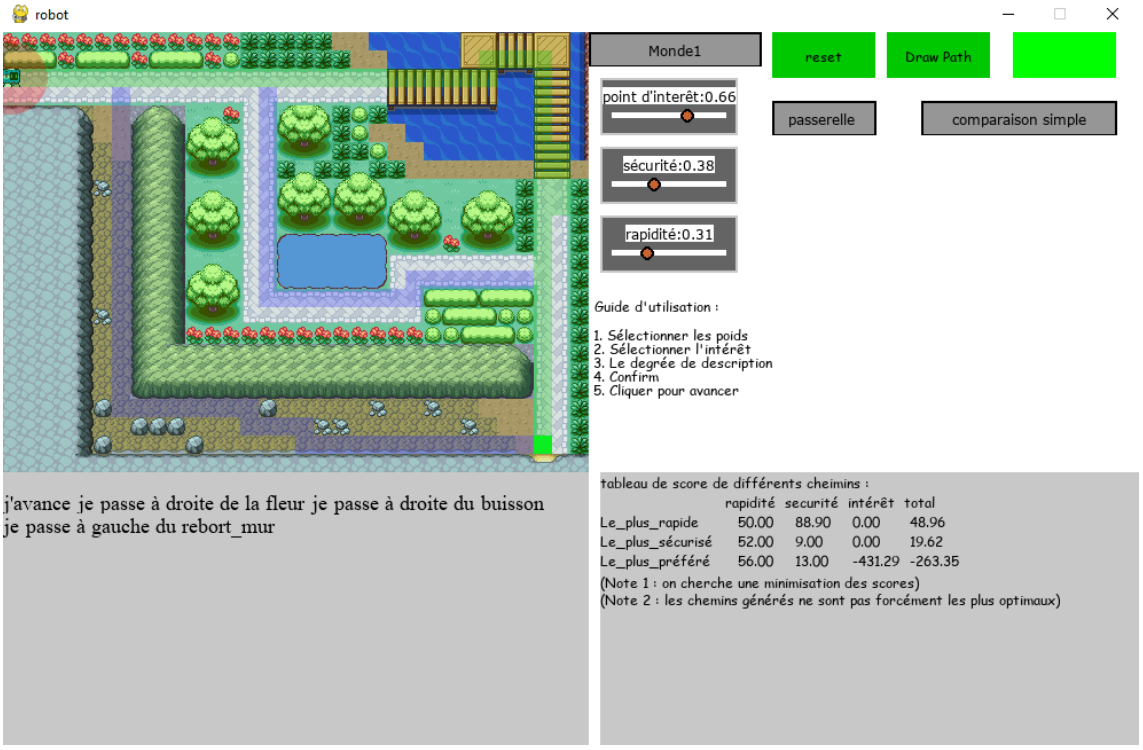


FIGURE 8 – Version final de notre fenêtre d'application

Nous avons implémenté trois degrés d'explication, une qui est une description simple, qui est la version où notre robot décrit de façon directe son environnement sans aucune explication, puis une comparaison simple qui est sélectionnée par défaut et qui en plus d'une description simple, il y a en plus des explications des informations essentielles aux intersections, et pour finir la dernière qui est une comparaison détaillée, il reprend les deux notions précédemment, mais comme son nom l'indique il explique dans les détails de façon qualitative et linguistique de tous les avantages et inconvénients du chemin qu'il compare en fonction des scores calculés et qui est affiché en bas à droite de la fenêtre.

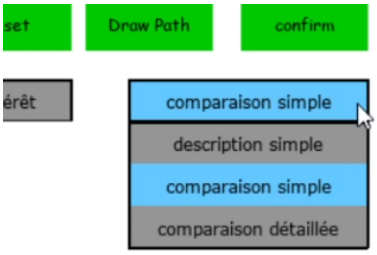


FIGURE 9 – Option du choix du degrée d'explication

FIGURE 10 – Description Simple

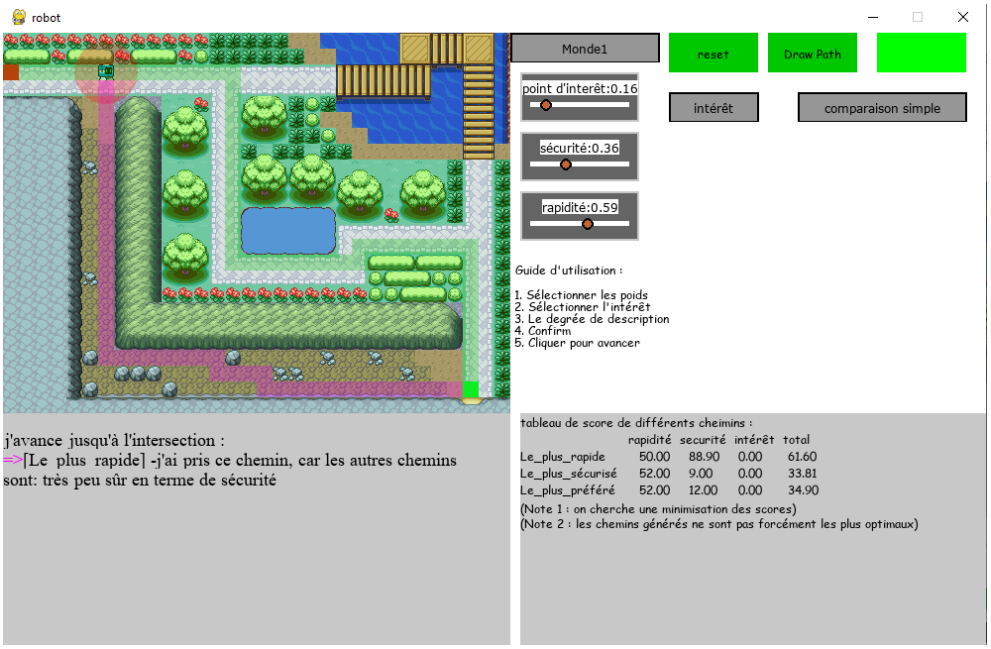


FIGURE 11 – Comparaison Simple

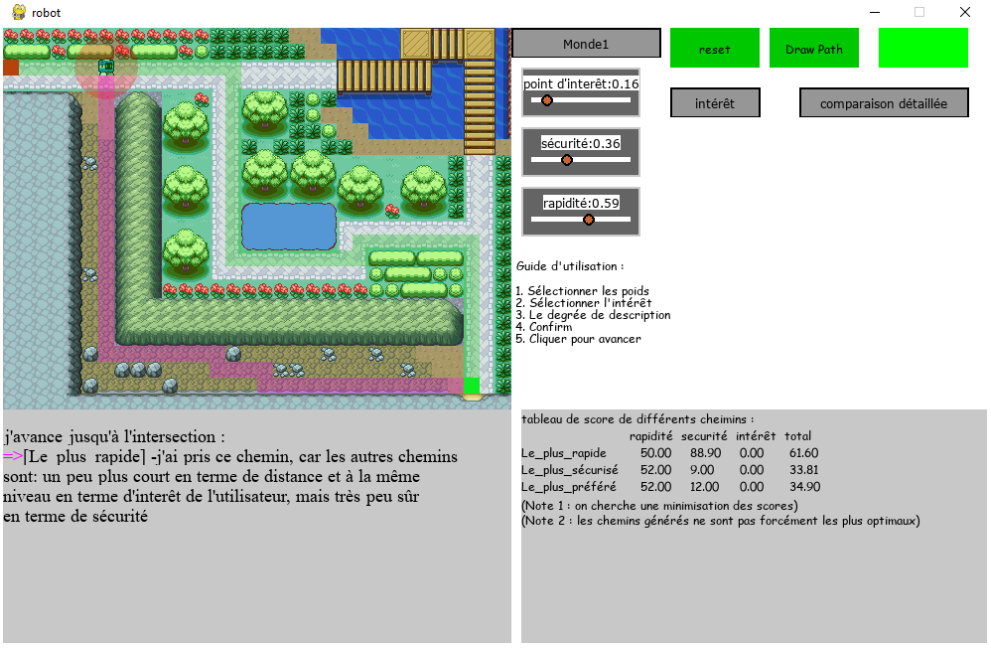


FIGURE 12 – Comparaison Détaillée

En plus de cela nous pouvons changer nos descripteurs qu'on a définis de base dans le fichier **MyEnum.py** pour changer le style des phrases qu'on veut y implémenter pour plus d'adaptabilité du projet, on a par exemple dans ce fichier 2 autres descriptions que vous pouvez commenter l'original et décommenter une nouvelle pour changer les phrases, ou directement changer l'original. Une version plus familière avec une notion d'émotion du robot est présente dedans, et une en anglais, mais il nécessite aussi qu'on change le fichier des descripteurs de propriété des objets, car ce fichier est en français et le nom des objets est en français.

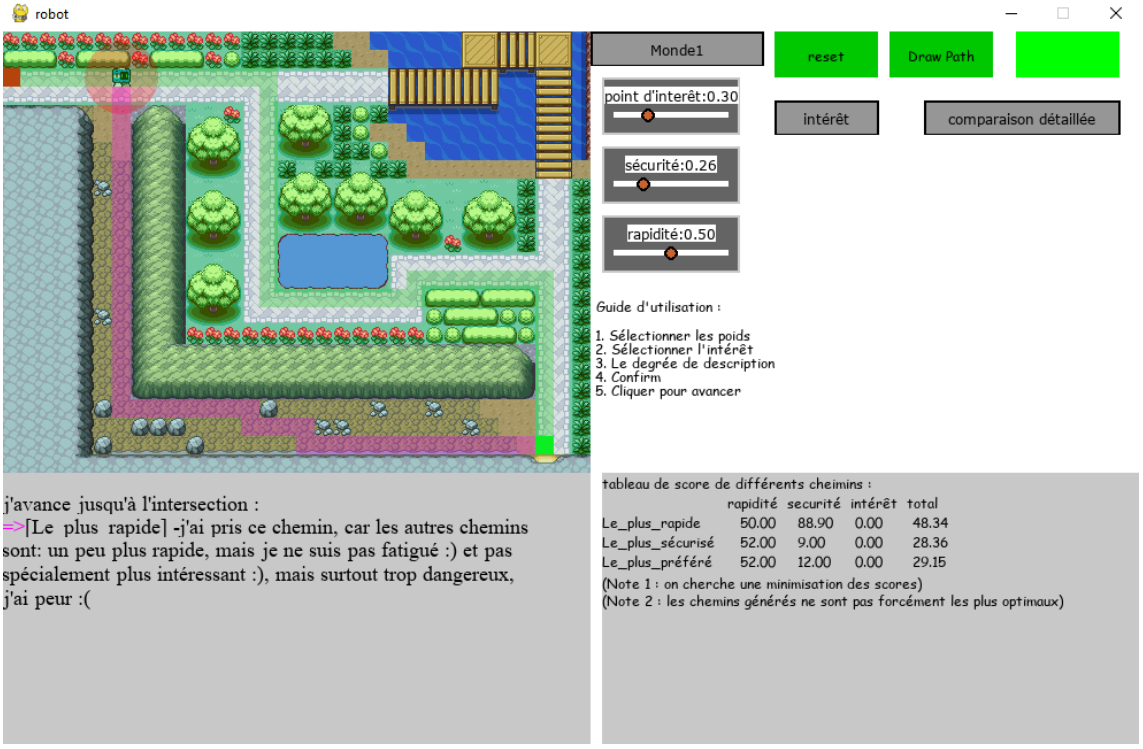


FIGURE 13 – Version plus Familier + Emotion du robot

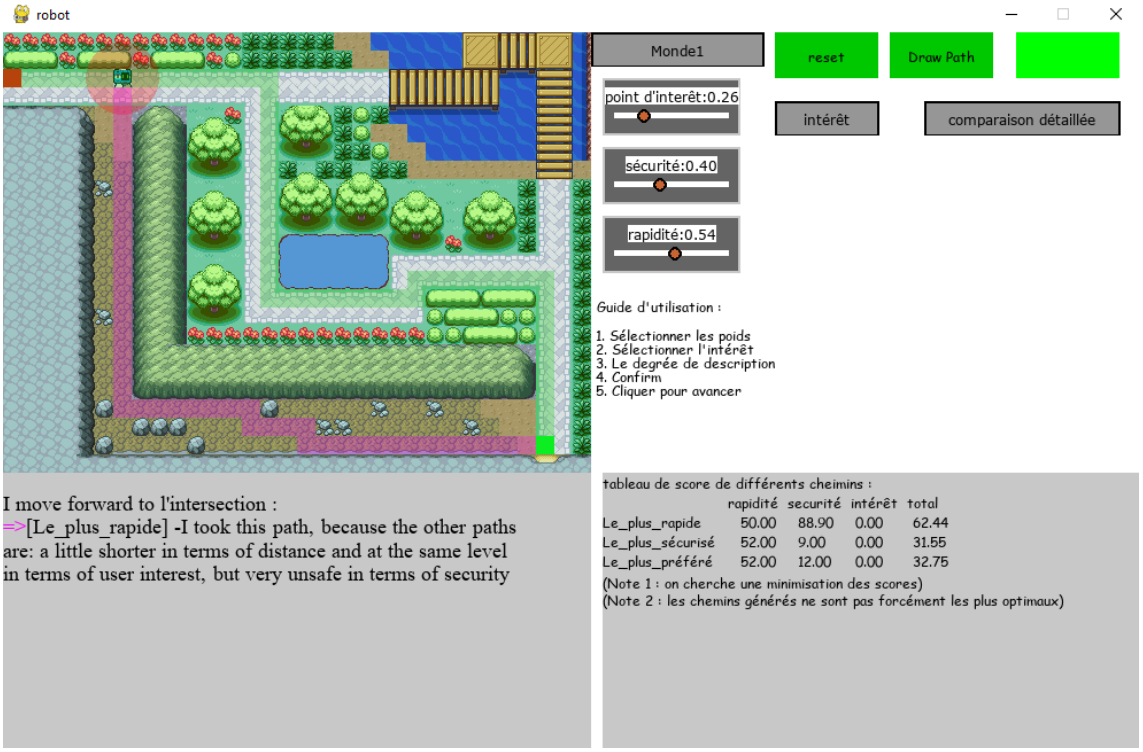


FIGURE 14 – Version anglais (version bêta)

Cette dernière version, en Anglais n'est pas encore bien fonctionnelle, car il nécessite des changements dans les propriétés du fichier qui décrit les propriétés des objets de la carte, en changeant leur propriété "name" en anglais et de quelque ajustement dans le code.

En résumé, notre application est assez flexible, mais qu'il a quand même ces limites comme avec la version anglaise, qui nécessiteraient des changements plus profonds dans le code, nous en parlerons de l'inconvénient de cette méthode dans la suite, mais il est toutefois une très bonne solution pour diversifier encore plus nos explications.

3 Géstion des chemins

3.1 Génération de chemin

Comme expliqué précédemment, nous avons utilisé un A* pour générer notre chemin.

Nous allons plus expliquer comment cela fonctionne à partir de maintenant.

3.1.1 A*

Le A* est un algorithme de plus court chemin qui permet de se rapprocher le plus possible en ligne droite d'un point A à un point B. les nouvelles cases sélectionnées dépend du coup jusqu'à arriver jusqu'à cette case et la distance qu'il reste à parcourir. La distance qu'on utilise est la distance de Manhattan (nombre de cases à parcourir sur un monde de type grillent).

3.1.2 Poids des cases

pour que nos chemins passent ou s'éloignent de zone, nous modifions les poids cases et ainsi le A* aura tendance à s'éloigner des cases qui ont des poids élevés (ces cases allongent la distance de l'arrivée) et se rapprocher des poids faibles (ces cases raccourcissent la distance à l'arrivée).

Les poids de danger se font sur une zone où on décrémente graduellement à la distance de l'objet donné (exemple poids de 5 à coter puis 4 à deux côtés etc.) tandis que celle de préférence est sur une zone d'un rayon de un de l'objet en poids négatif choisi arbitrairement à -7.

3.2 Découpe des chemin possible

Lorsqu'on donne des chemins possible, il est autorisé de faire plusieurs chemin qui se croisent (par exemple la figure 3 des multi-chemins). Nous avons pu gérer ceci grâce à un système assez simple qui est que pendant le parcours de chemin s'il y a un croisement, on copie les données de chemin déjà parcouru qui permet ne pas revenir en arrière et on sépare les chemins pour continuer la recherche. L'inconvénient de cette méthode et le coût en mémoire s'il y a plusieurs boucles. En effet, il y aura environ 2^n copie avec n, le nombre de boucles dans le chemin.

FIGURE 15 – Exemple de Draw Path a ne pas faire, au risque de faire planter

Cet exemple fait que le logiciel ne finira jamais. Car notre algorithme calculera toutes les chemins possibles sur les tracés que l'utilisateur a faits, et l'a complexité est exponentiel. Mais aussi d'évité de faire des "boucles", même si cela marcherait toujours mais l'algorithme ne le prendra pas en compte, car il cherche un plus court chemin et donc il ne va pas prendre un chemin s'il va revenir sur la même case par la suite, cette partie nécessite. Et donc le conseil serait de dessiner des chemins sans ambiguïté, car notre robot ne va pas suivre le tracé fait par l'utilisateur bêtement, mais il va calculer sur les cases où l'utilisateur a tracé le meilleur chemin possible en fonction des poids donnés.

4 Explication Linguistique

Notre partie explication linguistique se construit en 3 étapes, pour avoir un code le plus générique possible, et ainsi réutilisable avec plusieurs descriptions possibles.

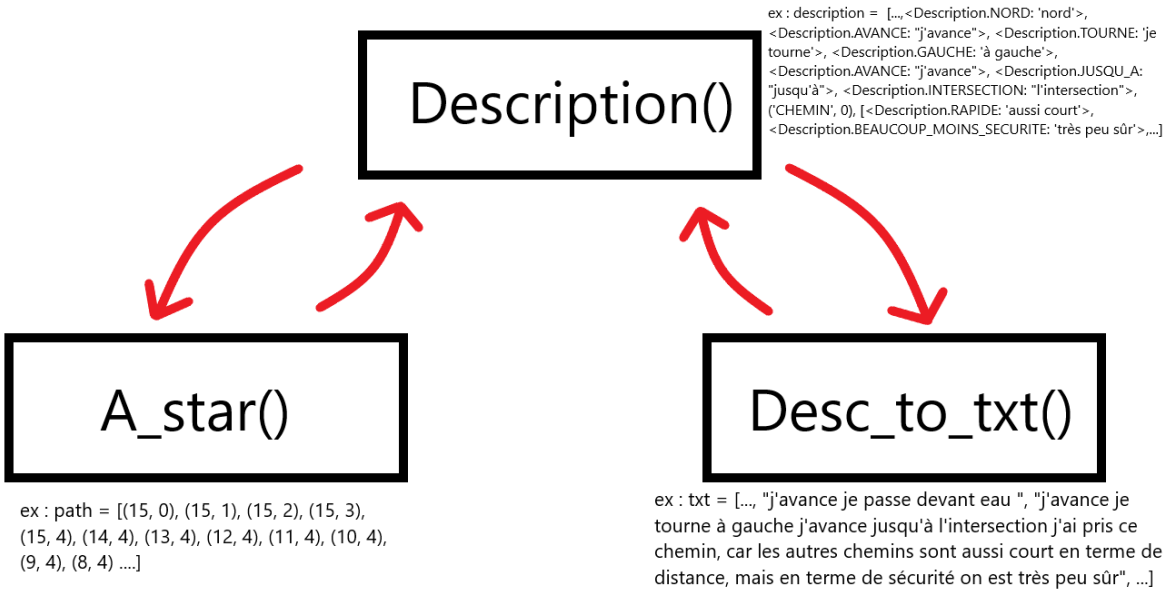


FIGURE 16 – Schéma de transition de la transformation de l'information

Avant de commencer le processus de transformation en message linguistique, il nous faut tout d'abord un fichier .xml, définissant notre monde, par exemple les objets présent sur notre carte, les caractéristiques de ces objets (niveau de dangerosité, nom féminin/masculin, possibilité d'accès sur la case de l'objet...). Toutes ces informations sont décrites dans notre fichier **ressource/descripteur.tsx** (.tsx est une variante d'un fichier .xml, qui est générée automatiquement à l'aide de Tiled), et ainsi que des cartes de type .tmx qui sont générées avec Tiled.

4.1 Explication Linguistique : A*

La partie **A*** a été expliquée dans la partie précédente, mais le concept primordial ici, c'est le fait qu'il nous renvoie une liste de coordonnées des différents chemins, pour ainsi les traités plus tard dans la partie descriptions.

4.2 Explication Linguistique : Description

Après avoir reçu les différents chemins en entrée et ainsi que le dictionnaire de description qui comporte toutes les caractéristiques de tous les objets sur le monde, notre fonction ici construite petit à petit une liste de description en rajoutant des informations concrètes, en fonction de son environnement. Tout au long du chemin où le robot doit suivre, la fonction rajoute les éléments concrets, par exemple pour signaler qu'a l'itération i notre robot tourne à droite, notre fonction va détecter que le sens d'orientation va changer et ainsi rajouter dans la liste des descriptions concrètes du type :

- **Avancer** : list[i].add(**Description{Avancer}**)
- **Tourner** : list[i].add(**Description{Tourner}**)
- **(Objet, "fleur")** : list[i].add(**Description{RencontreObject}** + **Description{MessageFleur}**)

Où encore pour calculer la position dans son espace, pour savoir où se trouvent exactement les objets où notre robot passe à côté, notre fonction calcule en fonction d'où il est orienté et de la position où se trouve l'objet qui est dans le rayon d'interaction avec le robot, et ainsi il peut rajouter dans la liste des descriptions, qu'il est passé à droite/à gauche/derrière/devant un objet de façon précise.

C'est ainsi que toute la partie description simple est faite.

Pour la partie explication, il est ajouté à cette liste à chaque séparation/intersection entre notre chemin ou un autre chemin dessiner ou générer automatiquement, et il aura l'ajout des descriptions de message, pour signaler le degré d'importance d'un événement grâce à des seuils fixer dans le fichier **myEnum.py** avec toutes les traductions linguistiques des descriptions concrets. Ainsi, par exemple si le ratio de sécurité entre le chemin choisi et le chemin comparé est supérieur a une seule fixée il aura l'ajout dans la liste de description, l'ajout d'un degré plus important, exemple :

- ratioSecuriter > 1.4 : list[i].add(**Description{MoinsSECURISER}**)
- ratioSecuriter > 3 : list[i].add(**Description{BeaucoupMoinsSECURISER}**)

On pourra alors définir dans notre fichier **myEnum.py**, les phrases qui correspondent à *Description{MoinsSECURISER}* et *Description{BeaucoupMoinsSECURISER}*.

Ainsi nous avons toutes les informations d'interaction avec l'environnement tout au long du trajet du robot, et il nous reste plus qu'à définir les messages linguistiques, pour traduire cette liste de description.

4.3 Explication Linguistique : Description à Texte

Une fois qu'on a notre liste de description avec toutes les informations sur chaque itération, on peut interpréter comme bon nous semble ces informations, par exemple, quant à l'itération x nous rencontrant plusieurs objets de type "fleur", alors nous pouvons envoyer un message avec une notion de quantité, ou encore remarquer dans la liste qu'il y a une description concrète disant qu'il y a une fleur à droite et à gauche, alors nous pouvons utiliser des notions de position plus précise et afficher un message linguistique "on passe entre...".

La version proposée dans le fichier **Traduction.py**, parcours toutes la liste des descriptions et il récupère les informations voulues dans la liste et les stocks pour ensuite les utiliser, par exemple a l'itération x, la fonction va parcourir tout sur ces sous liste et stocker tous les objets, et a la fin s'il détecte qu'il y a plusieurs descriptions pour dire qu'il y a une "fleur" a sa droite, il ne va l'afficher qu'une fois, ou encore s'il détecte qu'il a une "fleur" à droite et à gauche alors il va afficher qu'il passe entre les fleurs, et ainsi faire les bons accords pour que linguistiquement ça soit correct.

Pour les explications de choix de chemin, on a stocké comme avec les objets tous les messages d'explication (exemple : le chemin X est plus rapide et plus de points d'intérêt, mais il est très dangereux), pour ensuite afficher d'abord les messages "positifs" du chemin comparé, pour ensuite affichée "..,mais.." avec le ou les messages négatifs de pourquoi on a pris le chemin sélectionné.

En résumé notre code reste très flexible et adaptable et nous avons fait aussi la possibilité de changer directement les types de messages qu'on veut ou changer de langue comme expliquer dans le scénario 5. Ainsi dans **myEnum.py** on peut changer les hyperparamètres assez facilement.

5 Poursuite du Projet

Nous avons ajouté beaucoup de fonctionnalités sur notre fenêtre d'affichage, que vous pouvez découvrir dans la partie "Scénario 5". Et on s'est rendu compte qu'une fois, après avoir réussi à générer des chemins de différentes manières et d'avoir une base en description et explication linguistique, nous nous sommes rendu compte que nous avons toutes les bases pour faire toutes les explications que nous voulons, où enfaite chaque envie d'ajout d'une précision d'une explication n'était enfaite qu'une option qu'on rajoute lors de la phase de traduction de notre liste de description en message linguistique. Les points qu'on aurait voulu ajouter en plus seraient le fait, d'avoir tout d'abord une meilleure précision dans les différentes descriptions ou explications, mais qui revient a rajouté des optionnalités en plus, mais aussi une amélioration de la robustesse de notre code ou des approches différentes, ainsi comme exemple :

- Avoir une meilleure localisation des Objets dans l'espace, car notre robot ne peut distinguer un objet que dans les cas suivants à droite/à gauche/derrière/devant, par exemple si un objet est en diagonale, notre robot ne va pas distinguer si l'objet est à droite ou devant lui, il va par défaut privilégier les notions à droite et à gauche, une solution ici serait d'avoir une notion du degré où se trouvent l'objet et le robot pourra ainsi être beaucoup plus précis dans la description simple d'où se trouvent les objets, et ainsi rajouté des notions de positionnement distancié "très loin, loin, proche, très proche à X degrés(ou à quelle direction) de l'objet X"
- Nous avons suivi un model où notre robot explique pas à pas en fonction de l'avancer à chaque itération, mais nous aurons pu aussi avoir l'aspect séquentiel, qui était une des pistes qu'on pourrait poursuivre par la suite, exemple : "passer proche de l'objet A avant de passer très proche de l'objet B"
- Notre model reste toutefois asser robuste et il est assez maniable, par exemple dans le fichier **myEnum.py**, il y a plusieurs *class Description(Enum)*, par défaut on utilise le "Français Originel", mais on a voulu tester si en changeant les phrases notre algorithme marcherait toujours ou non, et les résultats sont assez convaincants, nous avons alors ajouté d'autre version , comme par exemple la version "Française familier, avec émotion et un petit smiley de l'état du robot", qui parle dans un français plus familier avec des phrases où notre robot peut exprimer ces émotions, exemple :

  "je n'aime pas"(pour exprimer le point d'intérêt du chemin),
  "je n'ai pas peur"(pour exprimer la dangerosité d'un chemin),
  "je ne suis pas fatigué"(pour exprimer la longueur du chemin).
  Ou encore une version anglaise.

Mais cela reste peu pratique, car cette méthode avec un Enum, n'est pas adapté pour pouvoir changer les descripteurs facilement, une solution serait ici de changé cette méthode Enum avec des fichiers XML pour avoir une meilleure adaptabilité des descripteurs qui sera interchangeable facilement.

— Et un dernier point qu'on aurait voulu aborder, mais impossible à réaliser à notre échelle, c'est le fait de donner un data set de différentes explications écrit par des humains sur différentes trajectoires, pour que notre robot puisse apprendre de lui-même comment procéder à une explication, grâce à tous les phrases écrites par des utilisateurs qu'on leur a demandé de faire une explication qualitative linguistiquement d'une trajectoire en particulier, pour ensuite faire apprendre avec un réseau de neurones récurrents, pour les traitements de langage et ainsi construire de lui-même une explication qualitative linguistique.

6 Conclusion

En conclusion, dans l'ensemble nous sommes très heureux d'avoir pu travailler sur ce projet. Nous avons plutôt bien aimé réfléchir sur un sujet qui était très ouvert et nous a permis d'explorer dans le domaine qu'on voulait. Nous avons vu à quel point programmer n'est qu'une petite partie du travail dans le domaine de la robotique. La recherche de solutions existantes fût très intéressante, mais qui pour la plupart n'est pas adaptée à notre sujet d'études, car peu avait réellement publié d'article à ce sujet, qui a première vue très basique mais très peu documenté. Réfléchir sur une première piste et partir de zéro a été le plus difficile pour nous, mais en même temps ceux qui nous ont laissés le plus de liberté possible dans nos choix futurs. Se rendre compte que nous n'avions pas envisagé tous les cas, était un plus pour nous, car cela nous mène a plus d'ouverture dans ce sujet.

Les résultats réalisés sont très encourageants pour notre part et nous sommes contents de ce que nous avons réussi à produire mais déçus d'avoir perdu un temps assez conséquent dans l'aspect "affichage", qui est certes très important, car c'est la partie qui expose tout notre travail, mais nécessitant tout un apprentissage aussi à lui seul, qui n'était pas prévue et voulue de base quand nous avons décidé de travailler sur ce sujet.

Nous sommes heureux d'avoir pu entrevoir que le monde de la robotique était vraiment très large, et l'avons abordée de cette façon avec nos encadrants. Nous nous souviendrons que nous avons eu des moments de désespoir lorsque l'affichage avec pygame plantait constamment au départ mais nous avons eu aussi des moments de bonheur lors du résultat final avec ces niveaux de précision apportée à l'affichage de nos résultats de notre rendu finale.

7 Manuel developpeur

7.1 readfile.py

Dans ce fichier, nous trouvons toutes les fonctions qui servent à lire des fichiers.

read_map_csv

```
def read_map_csv(filename):
```

Dans un premier temps, nous avions stocké nos cartes en csv avec tiled. Cette fonction reprenait les données de la carte sous forme d'un tableau panda.

read_map_tmx

```
read_map_tmx(filename):
```

Cette fonction prenait en paramètre le nom de la carte tmx créer grâce à tiled et renvoie une matrice représentant la grille. les valeur de la matrice sont les id de l'objet associer descripteur.

read_desc_xml

```
read_desc_xml(filename):
```

Même si le format des descripteurs sont en .tsx l'intérieur est bien du xml. Cette fonction va lire le descripteur et renvoyer le dictionnaire dont la clé et l'id de l'objet et la valeur un dictionnaire nom de l'attribut et valeur associé.

7.2 slider.py

Ce fichier contient nos deux boutons spéciaux.

OptionBox

Cette classe permet de créer nos boutons déroulant.

Slider

Cette classe permet de créer nos sliders.

7.3 tools.py

Les fichier tools.py comporte tous les fichiers qui vont aider au code principal.

7.4 main.py

Ce fichier est le fichier à lancer notre binaire.

7.5 PCCH.py

Le fichier contenant nos fonctions d'appel à nos algorithmes de recherche de plus court chemin.

A-star

L'algorithme que nous avons décidé d'utiliser est le A*.

7.6 descriptionTrajectoire.py

Ce fichier compte tout le code pour calculer la liste de description a partir des différents chemin généré avant.

descriptiontTrajectoirePlusExplication

Cette méthode est la méthode principale qui construit toute la liste de decription qu'on pourra l'utiliser pour le traduire dans le fichier Traduction.py

7.7 Traduction.py

Ce fichier permet de faire des traductions d'une liste de description en liste de message linguistique utilisé par le robot pour affiché les messages.

Description to Txt2

Cette méthode est la méthode principale qui construit toute la lsite des messages linguistique que le robot utilisera par la suite.

```
def a_start(debut, goal, x_max, y_max, wall, weight=dict()):
```

Les paramètres à donner sont la position de départ, de fin, les bornes de la carte, une liste de cases qui sont des murs et un dico si l'on souhaite rajouter des poids aux cases.

7.8 myEnum.py

Ce fichier est un fichier remplaçable qui contient une classe Description qui stock les messages associés à une action ou une description.

7.9 game.py

Là où est stocké notre fenêtre graphique.

La classe Game

Elle permet de faire notre fenêtre graphique et comporte plusieurs méthodes pour cela.

```
def on_init(self):
```

Cette méthode permet d'initialiser la fenêtre est de lancer la méthode de choix de carte.

8 Cahier des charges

Introduction

On vit dans un monde de plus en plus connecté et la présence des robots dans notre quotidien joue un rôle de plus en plus importante. Ces robots peuvent être représentés par exemple comme les robots aspirateur, robot d'aide à la personne ou robot de livraison, etc. Et donc la cohabitation entre l'humain et le robot est fondamentale. Ainsi, l'objectif de notre recherche est d'apporter une amélioration dans les multiples choix des trajectoires des robots, comme par exemple en fonction de l'environnement où le robot est en service, il est capable de choisir sur un multiple choix, tel que le chemin le plus court, le plus sûr, etc. Et de pouvoir l'expliquer efficacement avec l'Homme.

On effectuera sur une méthode d'explication qualitative comparative justifiante pourquoi une trajectoire a été retenue plutôt qu'une autre et ces descriptions pourrons être obtenues à partir de calcul de positions relatives, par exemple le sens d'orientation de notre agent et ces points d'interaction avec son environnement ou des segments de la trajectoire, et des objets de l'environnement. Ces informations métriques seront ensuite converties en descriptions qualitatives, linguistiques de manière à ce qu'il soit compréhensible par la grande majorité, un exemple concret est le fait qu'un robot nous décrit linguistiquement le trajet qu'il a effectué et pourquoi il a pris ce chemin en particulier.

Spécification technique des besoins et des exigences

Contexte

Besoins

Nous avons rencontré nos enseignants monsieur Maudet, madame Beynier et madame Bloch, qui sont aussi chercheur/chercheuse à Sorbonne université au LIP6. Lors de leur recherche dans ce domaine, ils se sont rendu compte que les articles et recherche parlant de ce sujet sont très limités, et cela concorde avec notre propre recherche documentaire sur ce sujet, où il y a vraiment très peu de recherche dans ce domaine d'explication qualitative d'une trajectoire, par une méthode linguistique.
Notre travail est alors de faire un premier travail sur ce domaine, pour voir les approches utilisées pour répondre à notre sujet.

Objectifs

Notre objectif est de concevoir une stratégie d'explication qualitative d'une trajectoire, et plusieurs solutions sont envisageables :

1. Charger une scène appartenant à une base existante, ou chargement d'une nouvelle image.
2. Entrer une trajectoire : par exemple, à la souris directement sur l'image, ou via un format donnant une séquence de positions.
3. Éditer les descripteurs linguistiques (par exemple à quel seuil est-on très proche).
4. Description linguistique d'une trajectoire existante.
5. Génération d'un ensemble de trajectoires sur la base d'une description.
6. Sélection selon les objectifs de l'utilisateur : le logiciel choisit parmi des trajectoires (spécifiées par l'utilisateur, ou générées par le logiciel).
7. Justification : après la sélection d'une trajectoire T, l'utilisateur peut sélectionner un autre T' dans l'ensemble de trajectoires, et le logiciel justifie pourquoi T a été préféré à T'.

Description

Fonctionnalités

Générateur de scènes :
Il s'agira de générer une scène en 2D, à partir d'images. On pourra commencer avec des objets constitués de formes simples (Lego ou cartons de couleurs différentes). Les objets sont décrits selon plusieurs critères (au minimum deux, par exemple on

peut facilement calculer la surface d'un objet, son contour et donc son périmètre (nombre de pixels du contour) et un indice de compacité ; ou encore la couleur, qui pourra traduire des caractéristiques comme la fragilité ou la température des objets, dans le monde réel).
En sortie, une grille, avec les objets identifiés à leurs différentes positions. Le format dépendra vraisemblablement des outils logiciels choisis (image, grille d'occupation, scène 2D compatible avec Tiled...).

Langage de description de trajectoires :

On définit un langage de description des trajectoires dans le monde envisagé, à l'aide de descripteurs qualitatifs. On peut envisager les éléments descriptifs suivants :

- Positionnement distanciel "[très loin, loin, proche, très proche] du carré vert"
- Positionnement relatif par rapport à deux objets : "passer entre l'objet vert et l'objet jaune"
- Positionnement relatif distance par rapport à deux : "passer plus près de l'objet rond que de l'objet carré"
- Aspect séquentiel : "passer proche de l'objet A avant de passer très proche de l'objet B" ponctuel

Générateur de trajectoires : Une trajectoire est un parcours entre un objet A et un objet B.

- Savoir générer le plus court chemin pour aller de A à B, passant éventuellement par un point C (revient à calculer deux trajectoires, de A à C et de C à B).
- Savoir générer une ou plusieurs trajectoires satisfaisant une description linguistique.

Langage de justification comparative : On définit ici des objectifs de l'utilisateur.
Mais aussi définir des degrés de satisfaction des critères, et les combiner pour pouvoir comparer deux trajectoires et dire laquelle est la meilleure au sens des critères définis.

Différentes solutions

Comme dit plutôt dans l'introduction, la recherche dans ce domaine était très limité, mais on a pu quand même trouver des articles scientifiques ou de thèse qui nous a

permis de mieux nous renseigner dans des sujets similaires, où l'on pourrait utiliser les approches utilisé.

À noter : les liens des sources se trouvent en fin de page.

Évaluations des sources et extraditions des éléments qui nous intéressent

Évaluation de la source 1

[?] Planification de trajectoire pour la manipulation d'objets et l'interaction Homme-robot La source, publiée le mardi 5 juillet 2011, est une thèse en vue d'obtention du doctorat de l'université de Toulouse, délivré par Université Toulouse III Paul Sabatier, spécialité informatique et robotique. La source est disponible en ligne via le site HAL. HAL est une plateforme en ligne développée en 2001 par le Centre pour la communication scientifique directe (CCSD) du CNRS, destinée au dépôt et à la diffusion d'articles de chercheurs publiés ou non, et de thèses, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés. L'auteur de thèse, Xavier Broquère, est docteur du LAAS-CNRS. Il est ingénieur logiciel chez Renault Software Factory. Le travail présenté dans ce manuscrit porte sur le problème de la génération de trajectoire pour les robots manipulateurs dans un contexte d'interaction Homme-Robot. Les parties ressource, où il parle de la génération de trajectoire sous contraintes cinématiques, approximation de trajectoire, planification et exécution de mouvement pour l'interaction Homme-Robot, nous donnent les idées de base adaptées à notre projet. Il nous donne des notions de contraintes qu'on peut ajouter pour la trajectoire du robot dans notre projet.

Évaluation de la source 2

[?] Analyse qualitative de robots
La source, publiée le 16 novembre 2017 et qui n'a pas été mise à jour depuis, est un mémoire présenté en vue de l'obtention du grade de Docteur de l'Université d'Angers et est disponible en ligne via le serveur TEL (thèses-en-ligne). L'auteur de la thèse, Romain BENOIT, est un doctorant qui a fait sa thèse au Laboratoire Angevin de Recherche en Ingénierie des Systèmes. L'auteur n'a pas publié d'autre article depuis, et il était sous son directeur de thèse Philippe WENGER, qui est le Directeurs de recherche du CNRS qui est lui cité dans plus 5 908 fois dans des articles dans le domaine de l'ingénierie, l'auteur semble être un jeune débutant, mais sous la direction d'un directeur très qualifié. L'article présente une vision des applications robotiques et, plus particulièrement, vers la classification de manipulateurs, mélangeant algorithmes et méthodologies de plusieurs théories mathématiques. La source véhicule

une information très détaillée, dont la qualité est largement suffisante, voire assez complexe. Le travail présenté est bien référencé et contient jusqu'à 39 références bibliographiques, ce qui semble correct et nous offre donc la possibilité d'enrichir nos connaissances en proposant d'autres travaux similaires. Enfin, ce document a été choisi, car plusieurs des notions utilisées sont utiles pour nos recherches, tel que la notion d'analyse qualitative qu'aborde la mémoire, de manière très mathématique telle que les notions d'espace opérationnel qu'est un espace vectorielle de position et d'orientation de l'effecteur, qui sera très utile comme base pour la description des paramètres de base de notre robot, et pour ainsi nous guider dans la conception et de l'implémentation de notre solution.

Évaluation de la source 3

[?] Learning the Correct Robot Trajectory in Real-Time from Physical Human Interactions La source, publiée en décembre 2019, est disponible en ligne sur le site de l'ACM. Les auteurs sont Dylan P. Losey qui est professeur assistant au département de génie mécanique de Virginia Tech et Marcia K. O'Malley, professeur en génie mécanique de l'université de Rice. Ils ont déjà écrit plusieurs articles sur le domaine de l'apprentissage en robotique. Cet article porte sur les stratégies d'apprentissage et de contrôle qui permettent aux robots d'exploiter les interventions humaines pour mettre à jour en temps réel leur trajectoire et leur objectif lors de tâches autonomes. Il est expliqué de manière concrète comment ils ont fait évoluer leur système d'apprentissage. Nous pouvons critiquer certains résultats. En effet, certains de ces résultats portent sur un seul essaie, il aurait été plus pertinent de faire une moyenne. Cependant, les résultats les plus importants qui utilisent les outils précédents sont eux liés à plusieurs essais ce qui nous rassure sur les résultats finaux. Ce document pourra être une source d'idées lors de l'ajout d'obstacle mouvant, pour ainsi nous guider dans la partie interaction de notre solution.

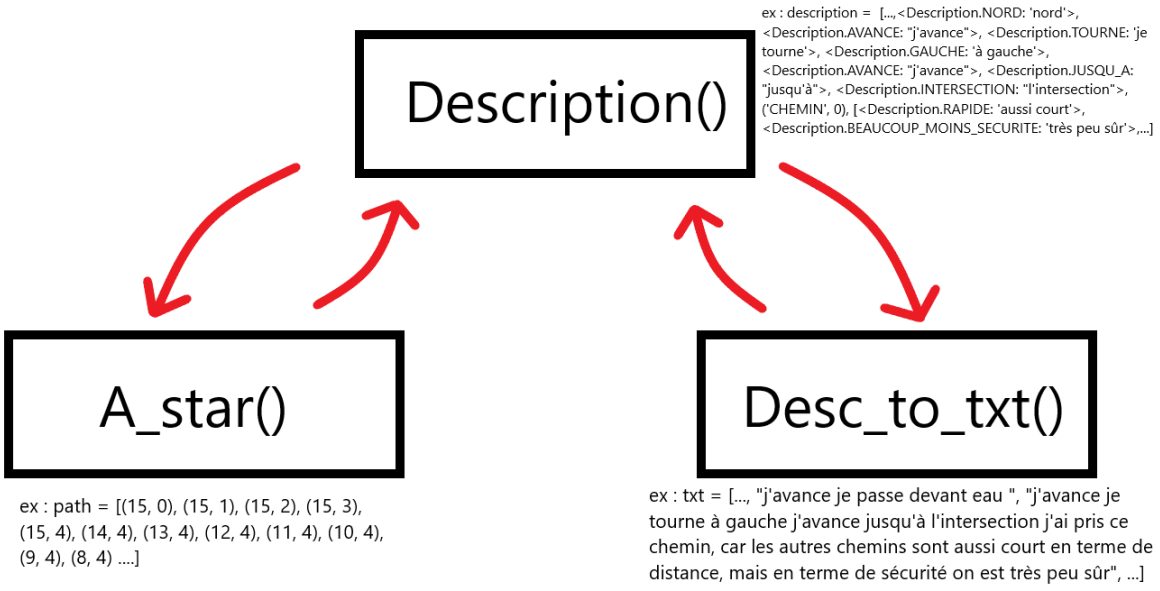
Notre solution

Nous nous orientons vers une l'implémentassions à partir de différents outils présents dans l'environnement python, tel que Tiled pour générer manuellement nos différentes cartes en 2D et travaillé directement dessus pour étudier chaque cas.

Puis d'implémenté nous-même notre génération de chemin, à partir d'un point de départ à un point d'arriver, on a fait le choix d'implémenter nous-même un A* (algorithme du plus court chemin), pour après intégrer des poids différent pour générer des chemins différant.

On utilisera Pygame, comme environnement d'affichage, car l'environnement est assez fourni et simple d'utilisation.

Et pour finir le plus important, pour la partie description et explication de trajectoire, on utilisera le chemin généré par notre A*, en la traduisant en une liste de description pour après le traduire en une chaîne de texte linguistique. Le tout sera affiché directement sur l'affichage Pygame.



Contraintes

Notre environnement est une surface plane et statique, représenté sous forme de grille. Notre robot occupe entièrement une case et a un rayon d'interaction définie au départ. Les objets occupent aussi entièrement une case, mais il peut y avoir plusieurs objet pour représenter un même objet, le robot ne peut aller que sur les cases dites "canPass" définie dans un fichier XML/TSX, par exemple un robot peut marcher sur une case nommé terre qui a cette attribut a "true", mais ne peut pas marcher sur des fleurs, car ils ont l'attribue a "false". Le robot connaît sa position initiale ainsi que le point d'arriver.
D'un point vue du niveau de l'interaction souhaité avec l'utilisateur on souhaite, que l'utilisateur puisse donner différent chemin possible, pour plusieurs comparaison de trajectoire possible, et aussi un certain de niveau de généricité des outils et fonction généré pour avoir un projet le plus maniable possible sur plusieurs points, comme par exemple changer le style linguistique ou le degré de précision (exemple : "très

proche", "un peu moins proche" ...)

Conditions de validation du projet

Scénario valant acceptation du travail

En accord avec nos trois encadrant, nous avons fixé des objectifs à atteindre au fur et à mesure des semaines. Un premier scénario choisi est de faire une description simple d'un trajet généré arbitrairement. Par description simple, nous entendons de pouvoir décrire très simplement tous les objets où notre robot passe à côté, exemple : je(robot) passe à côté des fleurs, je tourne à droite, j'avance, etc.

Puis nous devrons attaquer la partie explication, où l'on doit expliquer pourquoi un chemin est choisi par rapport à un autre, mais avant cela, nous devons générer notre carte graphiquement à l'aide de pygame, et d'utilisé des algorithmes de génération de chemin.

Livraison du code

Le code sera sur un github où tous les fichiers seront visibles et accessibles.

Livraison de la documentation

Pour la documentation, nous créerons une documentation par fichier de code et un guide d'utilisation. Tout cela sera intégré au rapport qui sera également disponible sur le répertoire du projet.

Planning de livraison

Rendu final pour le 27 avril 2021

Références

- [1] X. Broquère, "Planification de trajectoire pour la manipulation d'objets etl'interaction Homme-robot," phdthesis, Université Paul Sabatier - Toulouse III,Jul. 2011. [Online]. Available : https ://tel.archives-ouvertes.fr/tel-00644776
- [2] R. Benoit, "Analyse qualitative de robots," phdthesis, Université d'Angers, Nov.2017. [Online]. Available : https ://tel.archives-ouvertes.fr/tel-01760305

- [3] D. P. Losey and M. K. O'Malley, "Learning the Correct Robot Trajectoryin Real-Time from Physical Human Interactions,"ACM Transactions onHuman-Robot Interaction, vol. 9, no. 1, pp. 1 :1–1 :19, Dec. 2019. [Online].Available : http ://-doi.org/10.1145/3354139