REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed Khider – BISKRA

Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie

Département d'informatique

N° d'ordre : …… /M2/2020

Mémoire

présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : Images et vie artificielle

Recherche de chemin et optimisation multi-objectifs

Par :

OUAMANE MOHAMED WAIL

Soutenu le ……., devant le jury composé de :

Nom et prénom	Grade	Président
Bouguetitiche amina	MAA	Rapporteur
Nom et prénom	Grade	Examinateur

TABLE DES MATIÈRES

TABLE DES FIGURES

Abstract

The search of the path in one of the most important topics , there are many domains ; such as video games' domain , pedestrian traffic simulation's domain and other applications. One of the search's algorithms is used in the process of finding path .

Most of finding path's algorithms take into consideration only the distance when calculating the shortest path , but the researchers have noticed that the distance alone is not sufficient in this finding path process. There are other factors taken.

The goal of our work is to make the combination between the human being's standards when choosing his path and we apply this on a virtual agent. To realize this, we have used meshes navigation to represent the navigable surface and obstructions, then we have used the process of path's calculations by A-Star algorithm. We have taken into consideration other factors such as the crossing's cost of each area, deviation angle and number of direction change.

Résumé

La recherche du chemin est l'un des axes de recherche les plus importants, il existe dans plusieurs domaines, comme le domaine des jeux vidéo, le domaine de la simulation de le trafic des piétons (simulation de foules) et bien d'autres applications. L'un des algorithmes de la recherche est utilisé dans le processus de recherche de chemin.

La plupart des algorithmes de la recherche de chemin prennent en considération que la distance lors de calcule du plus court chemin. Mais les chercheurs ont noté que la distance seule n'est pas suffisante dans le processus de recherche du chemin ; d'autres facteurs sont pris en considération.

Le but de notre travail est de faire une combinaison des critères pris en considération par l'être humain lors du choix de son chemin et on applique cela sur un agent virtuel. Pour réaliser cela, nous avons utilisé le maillage de navigation pour la représentation de la surface navigable et les obstacles, puis nous avons exploité le processus des calculs de chemin à travers l'algorithme A étoile. On a pris en considération d'autres critères a part la distance comme le coût de la traversée de chaque zone, angle de déviation, et le nombre de changement de direction

Introduction générale

La recherche de chemin est un problème très bien étudié dans le domaine de l'informatique et il a de nombreuses applications dans le monde réel, telles que la détermination de l'itinéraire réseau le plus court, la navigation robotisée autonome, les jeux vidéo, l'animation comportementale, la détection du chemin le plus court entre la source et la destination sur une carte, etc. L'environnement étant un graphe, alors tous les problèmes de recherche de chemin abordent la question de savoir comment atteindre un nœud de destination à partir d'un nœud de départ dans un graphe. Cela peut être fait en implémentant un algorithme de parcours de graphe qui recherche dans ce dernier en commençant à un nœud arbitraire et en explorant les sommets adjacents des nœuds visités jusqu'à ce qu'il atteigne le nœud de destination.

Le problème de trouver un chemin entre deux nœuds dans un graphe n'est qu'une des deux principales questions auxquelles le chercheur de chemin tente de répondre. Une autre question qui peut répondre au problème de recherche de chemin est de savoir comment déterminer le chemin optimal entre le nœud parent et le nœud de destination.

Différentes stratégies sont appliquées pour résoudre le problème de recherche de chemin optimal tel comme utilisant la technique gloutonne dans l'algorithme de Dijkstra, la technique de programmation dynamique dans l'algorithme de Bellman-Ford, Floyd-Warshall, l'utilisation d'heuristique dans l'algorithme de recherche A *, etc.

-Problématique : Dans la plupart des cas, les algorithmes dédiés aux problèmes de recherche de chemin partent de l'hypothèse inhérente qu'il n'y a qu'un seul objectif que le logiciel tente d'atteindre, par exemple, minimiser la distance totale à parcourir ou réduire le temps nécessaire pour voyager d'un point à l'autre. On considère également que l'environnement sera pleinement connu à l'avance. Cependant, dans les applications du monde réel, nous travaillons fréquemment avec des environnements initialement inconnus. En outre, il est plus probable qu'au lieu d'un objectif unique, le problème à résoudre doit prendre en compte plusieurs objectifs contradictoires.

-Objectif : proposer une solution pour faire une combinaison entre les différents objectifs (distance, coût de la traversée des zones, angle de déviation, l'effort cognitif). Pour atteindre l'objectif que nous nous sommes assigné, nous avons organisé notre mémoire selon la structure suivante :

Le premier chapitre est un chapitre introductif sur l'animation comportementale et les modèles de comportement, les domaines d'application,

Le deuxième chapitre, il aborde l'état de l'art des algorithmes de recherche de chemin, quelques notions sur la théorie des graphes et les notions de l'optimisation multi-objectif.

La conception de notre application fera l'objet du troisième chapitre. Celui-ci explique la conception globale et détaillée de notre système, ainsi que chaque composante constituant le système.

Dans le dernier chapitre, nous présentons l'implémentation et la réalisation de notre travail.

Enfin, nous terminons ce mémoire par une conclusion générale de notre projet et les

perspectives.

CHAPITRE 1

ANIMATION COMPORTEMENTALE

Introduction

Les progrès de la modélisation et du rendu maintenant possible la représentation de mondes virtuel de taille et de complexité conséquente. Les domaines d'application de ces technologies s'étendent de la conception industrielle à l'aide à la formation en passant par les loisirs numériques et le cinéma. Certain acteurs des domaines de l'urbanisme et de l'architecture utilisent ces technologies pour simuler des foules dans des maquettes virtuelles. Ces industrielle souhaitent pouvoir peupler leur environnements virtuels d'humanoïdes dont le comportement serait plausible. À ceci s'ajoute souvent la nécessité de faire évoluer cet univers virtuel en temps-réel et de le rendre interactif. La complexité des scènes, le nombre des humanoïdes et surtout l'interactivité ne nous permettent plus d'animer à la mais. L'animation comportementale regroupe les recherches s'intéressant à l'autonomie de tels humanoïdes virtuels avec pour objectif d'automatiser le création d'animation crédibles du point de vue du comportement mais aussi, dans certains cas, de simuler les comportements humains [1].

Définition

L'animation comportementale cherche à offrir des modèles et des outils permettant la création d'entités virtuelles autonomes. Ces entités perçoivent leur environnement, agissent sur ce dernier et surtout prennent d'elles-mêmes des décisions en rapport avec la situation perçue dans le but d'exhiber un comportement cohérent proche de l'organisme vivant simulé [2].

Domaine d'application

1 Jeux vidéo

Avec l'évolution de la puissance de calcul des ordinateurs, allant de paire avec la puissance des cartes graphiques, une bonne qualité de rendu graphique peut désormais être obtenue en utilisant relativement peu de temps processeur. Cette évolution permet aux concepteurs de jeux vidéo de se concentrer sur le comportement des personnages avec lesquels les joueurs peuvent interagir. Deux exemples notables sont les jeux de créatures de la société Cyberlife, et Black and White, de la société Lionhead, qui proposent des mondes peuplés de créatures autonomes. Ces créatures peuvent prendre des décisions seules, l'intervention du joueur, qui peut cependant, dans certaines circonstances influer sur leur comportement est de tenter de leur apprendre des choses. Le développement de l'animation comportementale a fait naître une nouvelle tendance : la fiction interactive. L'idée est, ici, de fournir les grandes lignes d'un scénario, tout en laissant le joueur libre d'interagir à son gré avec des entités semi-autonomes. Ces entités suivent la ligne directrice du scénario mais peuvent prendre des décisions pour interagir «intelligemment» avec le joueur. Cela donne une grande sensation de liberté, tout en pouvant générer des situations imprévues faces aux actions de l'utilisateur. Dans le domaine de jeu, outre la qualité des animations et des graphiques, voir la figure (1.1). La crédibilité et la cohérence du comportement sont primordiales pour le réalisme et l'implication du joueur.

FIGURE 1.1 – Environnement virtuel des jeux vidéo.

2 Films et effets spéciaux

L'animation comportementale a fait son apparition dans le monde de la cinématographie (figure (1.2)) et des effets spéciaux depuis déjà un certain temps. La tendance actuelle est d'accroître son utilisation, à travers le développement d'outils de modélisation du comportement pour les logiciels de production d'animation en 3D. Cette tendance s'explique par

un besoin de productivité accru, à travers des scènes de plus en plus complexes mais allant de paire avec des exigences de qualité croissantes. L'un des meilleurs exemples actuels est le deuxième volet de l'adaptation cinématographique du seigneur des anneaux (figure (1.2)). Pour la grande bataille du gouffre de Helm, un outil spécifique : MASSIVE, a été utilisé pour reproduire les comportements de foule et modéliser la composante décisionnelle des entités. Dans le même domaine, divers outils sont désormais disponibles, parmi lesquels AI-Implant de la société BIOGRAPHICS, qui peut être utilisé sous la forme de module pour 3D studio Max et Maya, ou bien encore les modules RTK-Crowd et RTK-Behavior de SOFTIMAGE. En termes de conception, ces outils offrent de grands avantages :

— Le travail du concepteur est simplifié par la gestion automatisée des comportements de groupe.
— Les modèles décisionnels permettent de générer automatiquement des animations adaptées au contexte.
— Les scènes paraissent plus réelles car elles sont plus variées. Grâce à l'automatisation, ces techniques permettent de décrire, de manière implicite, des scènes d'une complexité inégalable grâce à l'aide d'outils dédiés

FIGURE 1.2 – Film d'animation.

3 Validité ergonomique des sites

La création de lieux publics pose des problèmes d'ordre ergonomique, autrement dit, relatifs à la qualité de leur utilisation. Des problèmes peuvent se poser quant à la bonne navigation à l'intérieur des lieux, la lisibilité des divers panneaux de direction, ou lors de situations de panique. L'animation comportementale, à travers des modèles se basant sur l'analyse du comportement humain, peut être utilisée pour effectuer des validations sur des maquettes virtuelles. Les éventuels problèmes peuvent alors être détectés et corrigés avant la construction des divers aménagements. Dans ce cadre, son utilisation offre des atouts qui sont d'ordres sécuritaire et ergonomique mais aussi d'ordre économique (figure(1.3)).

FIGURE 1.3 – La simulation d'entraînement Fire fighter Training.

4 Mise en situation

L'interaction avec des agents autonomes, à travers la réalité virtuelle, permet de mettre un être humain en situation dans le cadre d'un scénario. Ces capacités peuvent être utilisées dans un cadre pédagogique où l'interaction avec des humanoïdes intelligents aura pour conséquences d'augmenter la rapidité d'apprentissage par l'intermédiaire de moyens d'interaction plus proches de la réalité. Un autre domaine d'application concerne l'armée où la définition du comportement d'humanoïdes permet de tester des scénarios catastrophes ou de mettre les soldats en situation. Dans le domaine hospitalier, l'animation comportementale, couplée à la réalité virtuelle, peut être utilisée dans le cadre de soins aux personnes phobiques. L'idée consiste à plonger la personne, par l'utilisation de la réalité virtuelle, dans un milieu qui déclenche sa phobie. Dans le cadre du soin de l'agoraphobie, par exemple, les simulations de foules s'avèrent utiles, pour peupler les environnements virtuels (figure (1.4)) et fournir au patient un milieu réaliste.

FIGURE 1.4 – Visite virtuelle de la ville de Grenoble.

La boucle de l'animation comportementale

L'être humain évolue dans un environnement dynamique. Il est doté de trois capacités fondamentales lui permettant de réagir aux modifications de cet environnement et d'agir sur ce dernier : la perception, la décision, et l'action. Ces trois briques forment la boucle perception-décision-action qui constitue le modèle le plus général représentant un organisme vivant, autonome, évoluant dans un environnement dynamique. La figure 1.4 présente ce modèle, qui est à la base d'un grand nombre de théories relatives au comportement, dans le quel chaque brique possède un rôle bien défini [1] :

- **La perception** alimente le processus décisionnel avec des informations captées sur l'état de l'environnement. Ces informations sont locales ; un organisme, quel qu'il soit, ne peut percevoir que des informations qui se trouvent à proximité de lui. Bien entendu, cette notion de proximité comparativement à la taille du monde, peut être considérée comme ponctuelle
- **L'action** représente les moyens que l'organisme possède pour modifier l'état du monde qui l'entoure. Une action peut se traduire sous plusieurs formes : parler, manipuler un objet, se déplacer... .
- **La décision** joue un rôle central ; elle effectue une corrélation entre la perception et l'action. La perception alimente le processus décisionnel avec des informations « abstraites » issues des capteurs (ou sens) qui peuvent être mémorisées et/ou directement exploitées. A partir de ces informations, la composante décisionnelle choisit une ou plusieurs actions à réaliser qui sont adaptées au contexte et aux motivations de l'individu.

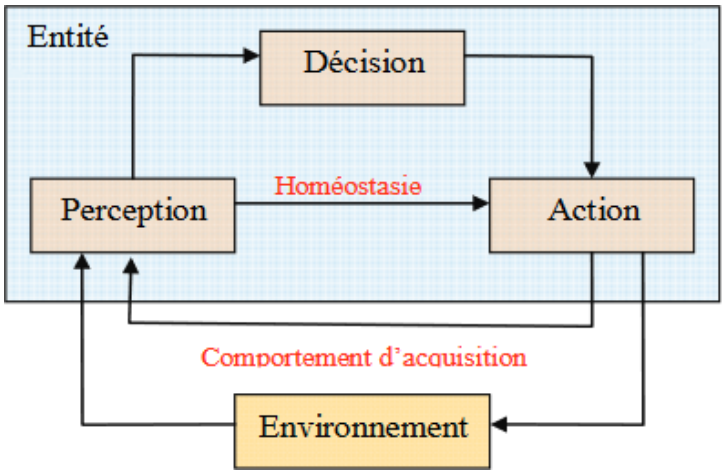


FIGURE 1.5 – la boucle d'animation comportementale

Modèles de comportements

Les processus décisionnels peuvent être classés en deux grandes catégories le premier est le modèle **réactifs** représentent des comportements pré-câblés ne faisant pas directement intervenir de processus conscient, le deuxième c'est le modèle **cognitifs** qui fait une représentation abstraite des connaissances et permettent de sélectionner une suite d'actions cohérentes visant à atteindre un but donné. L'objectif de ce section est de présenter les grandes techniques qui ont été proposées pour modéliser la sélection d'action pour les humains virtuels ou plus généralement pour les entités autonomes dans le domaine de l'informatique. Par similarité avec les comportements humains, ces techniques peuvent être classées dans deux catégories : les systèmes réactifs et les systèmes cognitifs et orientés but [3].

Modèles réactifs

Les systèmes réactifs permettent de modéliser des comportements plus ou moins complexes n'utilisant pas directement de modélisation abstraite des connaissances. Ces systèmes sont caractérisés par le fait qu'ils relient la perception à l'action sans effectuer de raisonnement nécessitant une projection dans le temps. Ces comportements, qualifiés de réactifs, peuvent être classés dans deux catégories : les comportements réflexes et les comportements spécialisés [3].

1. **Système stimuli-réponses** Les systèmes stimuli-réponses font partie des premières générations de modèles de comportements. Ils sont issus des travaux développés en éthologie (comportement des animaux) dans lesquels les comportements sont le résultat direct

d'une série d'interactions avec l'environnement. De façon générale, la représentation s'effectue sous la forme de réseaux de nœuds interconnectés. Les nœuds en entrée du réseau propagent l'information perçue, alors que les nœuds de sortie sont connectés aux effecteurs. L'idée est donc de traduire par l'intermédiaire de ce réseau une fonction mathématique permettant de corréler directement la perception à l'action[3].

2. Systèmes à base de règles

Le comportement des entités sont modélisés sous la forme d'un ensemble de règles. Le choix du comportement à adopter est alors défini par pré -condition sur l'environnement stipulant dans quel contexte adopter le comportement correspondant. Cette approche présuppose donc l'existence d'un ensemble de règles couvrant l'ensemble des situations possibles. Reynolds, pionnier dans le domaine de l'animation de nuées, propose un modèle décentralisé, basé sur ce modèle . Le comportement de chaque entité appartenant à la nuée est régi par trois règles : s'éloigner des voisins pour éviter les collisions, rester proche de ses voisins pour garder une cohésion et enfin réguler la vitesse de l'entité sur les entités voisines. Le respect de ces règles donne lieu à un comportement de groupe émergent. Les entités évoluent sous la forme d'une nuée se dirigeant vers un point cible qui peut être dynamiquement modifié. Dans cette approche, l'autonomie de l'entité est discutable dans la mesure ou son comportement est uniquement défini en fonction du comportement de son voisinage et pas en fonction d'une volonté propre [3].

(a) **Les systèmes à base de vote**. Des modules ayant chacun un objectif distinct notent les différentes règles, la règle élue est celle la mieux notée.

(b) **Les arbres de décision**. Chaque action élémentaire est représentée par une feuille de l'arbre (voir fig. 1.5).Pour savoir laquelle est élue, le choix s'opère au sein des nœuds.

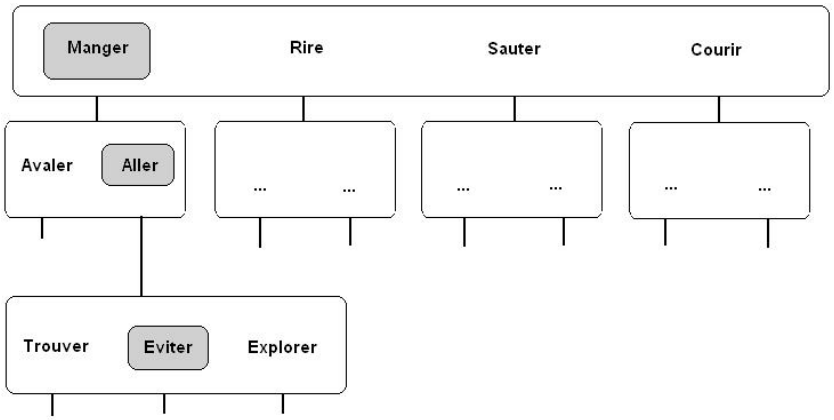


FIGURE 1.6 – Arbre de décision.

3. **Systèmes à base d'automate** La réalisation d'un comportement consiste en un enchainement d'u n certain nombre de taches considérées comme élémentaires à un certain niveau d'abstraction. Prenons l'exemple d'un comportement adopté lors d'un jeu de cache -cache par la personne devant se cacher. Dans un premier temps, la personne se cache puis attend. Deux cas peuvent alors se présenter : soit la personne est touchée durant sa phase d'attente et une nouvelle partie recommence, soit la personne se rend compte qu'elle est trouvée, auquel cas elle s'enfuit . . . Ici, les actions élémentaires sont des déplacements et des attentes. L'enchainement de ces actions dépend d'une part de la dernière action effectuée et d'autre part du contexte (touché, vu,. . .)[1].

   (a) **Les automates parallèles**. Dans ce système, un automate actif s'exécute tandis que les automates en attente sont empilés dans une pile en attendant leur tour.

   (b) **Les automates parallèles hiérarchiques**. Ici plusieurs automates s'exécutent en parallèles, ils peuvent ainsi gérer des actions simultanées.

   (c) **Les piles d'automates**. Ce système gère également le parallélisme entre automates mais instaure en plus une notion de hiérarchie, les pères étant vu comme une composition des automates fils .

Modèles cognitifs et orientés but

Les systèmes cognitifs et orientés buts nécessitent une représentation abstraite du monde afin de rechercher une suite d'actions permettant de satisfaire un but donné. Le comportement n'est donc plus complètement décrit mais bel et bien calculé. La recherche des actions permettant de satisfaire un but utilise une forme de projection dans le futur permettant de raisonner sur les conséquences des actions [3].

Le calcul situationnel

Il s'agit d'un formalisme largement utilisé dans le domaine de l'intelligence artificielle permettant de décrire des mondes très complexes ainsi que les actions qui peuvent être effectuées par des entités le peuplant. Cette approche est basée sur six concepts permettant de représenter le monde du raisonnement :

— **Les situations** représentent l'état complet du monde à un instant donné. Elles sont généralement décrites par un ensemble de faits.

— **Les fluents** décrivent une propriété dynamique du monde. Ces derniers sont définis comme des fonctions prenant une situation en paramètre renvoyant l'état de la propriété. Généralement, ils représentent un fait atomique (il pleut) ou encore une relation entre des objets du monde (le parapluie est dans le placard).

— **Les causalités** représentent des relations de cause à effet. Elles permettent de déduire l'état de certains fluents en fonction de l'état d'autres fluents.
— **Les actions** permettent de modifier une situation. Elles sont décrites par l'intermédiaire de pré-conditions et d'effets. La pré-condition d'une action se traduit par une expression booléenne utilisant les fluents. Elle qualifie un sous état de la situation qui doit être satisfait pour pouvoir exécuter l'action. Les effets de l'action décrivent la manière dont la situation est modifiée par l'action. Ces effets peuvent être inconditionnels (indépendants de la situation) ou conditionnels (dépendant de la situation) [3].
— Les stratégies représentent des enchaînements d'actions préconçus. Elles correspondent à des étapes de raisonnement scriptées.
— **La connaissance** permet de modéliser le fait que l'entité connaisse ou non l'état de certaines propriétés du monde. Par ce biais, il est possible de décrire des actions permettant d'acquérir de la connaissance (actions perceptives) et / ou dépendant du niveau de connaissance de l'entité sur l'état de l'environnement [3].

STRIPS : une version simplifiée de calcul situationnel

Le formalisme STRIPS (STandford Research Institute Planning System) exploite un sous ensemble du calcul situationnel permettant ainsi de proposer des algorithmes de planification plus efficaces. Le formalisme STRIPS s'avère moins expressif que le calcul situationnel. Cependant, cette restriction d'expressivité permet de proposer des algorithmes de planification plus performants en termes de temps de calcul et d'occupation mémoire. Ces modèles peuvent résoudre des problèmes complexes à partir du moment où ils s'expriment au sein du formalisme.

HTN : les réseaux de tâches hiérarchiques réseaux

La planification HTN (Hierarchical Tasks Networks) diffère des modes de planification présentés jusqu'alors. Plutôt que de décrire un ensemble d'actions atomiques permettant de modifier le monde et de chercher une suite linéaire d'actions satisfaisant un but,la planification par HTN fonctionne de manière hiérarchique en s'appuyant sur trois concepts : les tâches, les méthodes et les actions.

— **Les tâches** constituent ce que l'on peut qualifier de but dans les méthodes de planification. Elles acceptent de prendre un certain nombre de paramètres, rendant ainsi possible la description de tâches générales pour lesquelles les paramètres seront instanciés lors d'une demande de planification. Le pouvoir d'expression associé à la notion de tâche est donc très grand. Elles permettent notamment d'exprimer des comportements complexes n'ayant pas forcément une influence exprimable sous la forme de faits sur l'environne-

ment.

— **Les méthodes** Une méthode décrit une suite ordonnée de sous-tâches et/ou d'actions à réaliser pour remplir la tâche à laquelle elle correspond. Elle possède des pré-conditions décrivant l'état du monde nécessaire à sa réalisation. Une notion intéressante de la planification HTN apparaît alors : il est possible de décrire plusieurs méthodes permettant de réaliser la même tâche.

— **Les actions** sont représentées via le formalisme STRIPS, elles possèdent des préconditions et des effets. Elles sont directement réalisables, sous réserve de la véracité de leur pré-condition, sans avoir recours à une forme de décomposition.

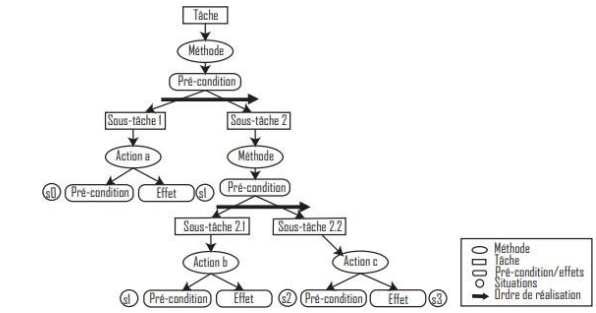


FIGURE 1.7 – Exemple de plan généré par planification au sein des HTN

La propriété de la planification HTN réside dans la mise à disposition des concepts de méthode et de tâche. D'une part, les tâches permettent d'exprimer des buts qui ne se traduisent pas forcément par un ensemble de faits [3].

Les mécanismes de sélection d'actions

Les mécanismes de sélection d'action ont été introduits pour gérer deux aspects du comportement situé : la réactivité et l'opportunisme face aux changements de l'environnement ainsi que la recherche d'une suite d'actions visant à satisfaire un but.

Les mécanismes de sélection d'actions présentent de très bonnes propriétés pour l'animation comportementale. Le monde est ici considéré comme dynamique et peut donc changer sans l'intervention de l'agent. Du point de vue de l'agent, ces changements sont pris en compte à chaque nouvelle sélection d'action, lui permettant de réagir rapidement tout en exploitant des opportunités offertes par l'environnement. Ces mécanismes permettent donc d'allier réactivité et planification. Cependant, ces systèmes souffrent de deux défauts. D'une part, ils peuvent osciller dans le choix des actions. D'autre part, ils ne sont pas complets. Autrement dit, même si un plan d'action existe pour satisfaire un but, rien ne garantit que ces systèmes puissent le trouver [3].

Les systèmes BDI

Le mode de raisonnement associé aux systèmes BDI (Beliefs Desire Intentions) s'inspire du raisonnement pratique de l'être humain

- Les croyances symbolisent la connaissance que l'agent possède de l'état du monde ou plus précisément ce que l'agent croit vrai. Elles sont caractérisées par une incomplétude due aux capacités de perception réduites de l'agent et à l'hypothèse de départ sur la dynamique du monde qui rend impossible la prédiction de son état.
- Les désirs représentent les motivations de l'agent. Cette notion peut être assimilée à un but mais ne possède pas une formalisation telle que dans le calcul situationnel ou le formalisme STRIPS. La notion de désir se rapproche de la notion de tâche dans le formalisme HTN.
- Les intentions regroupent l'ensemble des plans que l'agent adopte en vue de réaliser ses désirs [3].

Représentation de l'environnement

Dans le cas général, les graphistes et les architectes fournissent l'environnement sous la forme d'une base de données et trois dimensions. L'objectif est de gérer un déplacement cohérent des entités sur la base de ces informations géométriques. Pour cela, des représentations plus simples sont calculées, éliminant les informations superflues et rendant possible des calculs rapides. L'environnement est ramené à une carte en deux dimensions contenant des informations sur l'espace navigable, c'est à dire l'espace dans lequel l'humanoïde peut se déplacer. Sur la base de cette représentation, un chemin permettant à un humanoïde d'atteindre un point défini est planifié [**1**].

Représentation de l'espace libre

La notion d'espace navigable est reliée à la morphologie de l'humanoïde : sa taille, sa largeur, sa capacité à se baisser, à sauter. La détection d'obstacles est facilitée par l'utilisation d'un volume englobant de l'humanoïde (le plus souvent un cylindre) correspondant à l'espace nécessaire pour tenir debout. La planification a pour objectif de déplacer le volume englobant dans l'espace libre. Il existe deux grandes familles de techniques de représentation de l'environnement : **la décomposition en cellules** et **les cartes de cheminement** [**1**].

Décomposition en cellules

L'espace libre est découpé en zones, correspondant à des cellules. Les relations de proximité sont représentées grâce à un graphe. Ces décompositions peuvent êtres **exactes** ou **approximatives** [**1**].

Décompositions approximatives :

L'approche la plus immédiate consiste à discrétiser l'espace de manière uniforme, suivant des figures régulières, le plus souvent des carrés ou des rectangles. Une information est associée à chaque cellule : libre, occupée par un obstacle ou mixte (partiellement occupée par un obstacle). Une carte d'atteignabilité indique pour chaque cellule si elle est navigable ou non suivant la configuration des obstacles et de la morphologie de l'humanoïde [1].

La décomposition uniforme permet d'accéder rapidement à un point connaissant ses coordonnées. Cependant, elle souffre de certains défauts. Une cellule pouvant à la fois contenir un obstacle et une partie accessible, des informations utiles sont perdues. Ainsi, pour augmenter la précision, il faut diminuer la taille des cellules. En conséquence, la quantité de mémoire utilisée augmente de manière quadratique. Il est donc nécessaire de trouver un compromis sur la taille des cellules pour avoir une précision et une occupation mémoire satisfaisantes. L'utilisation de grilles hiérarchiques consiste à découper l'espace grâce à un arbre de cellules. Les cellules est entièrement libres ou entièrement occupées sont conservée telle quelle. En revanche, les cellules mixtes sont de nouveau divisées. Le processus s'arrête quand les cellules ont atteintes une taille minimale. Cette technique permet donc une compression des données puisqu'elle ne rajoute de l'information qu'aux endroits où c'est utile.

Décompositions exactes

Une décomposition exacte est obtenue en faisant coïncider les limites de l'espace navigable avec le bord des cellules. La triangulation de Delaunay contrainte divise l'espace en cellules triangulaires. Les arrêtes sont de deux types : contraintes si elles coïncident avec un obstacle, libres si elles sont obtenues par triangulation (Figure 1-8 (b)). Des nouveaux segments sont insérés au niveau des rétrécissements de l'espace pour mieux pouvoir détecter les zones inaccessibles (Figure 1-8 (d)) . Les cellules triangulaires sont ensuite regroupées en cellules convexes, en conservant les informations sur les rétrécissements. Pour finir, chaque cellule est associée au sommet d'un graphe, une arrête existant entre deux sommets s'il est possible de passer d'une cellule à l'autre [**1**].

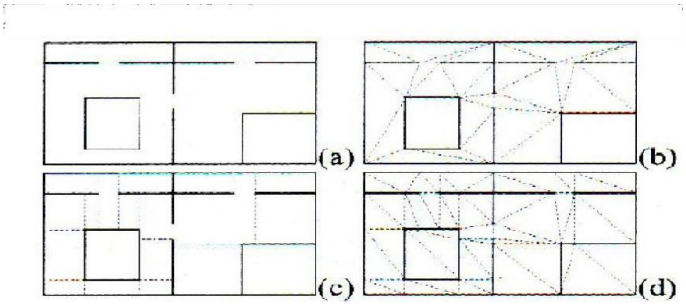


FIGURE 1.8 – Décompositions exactes

(a) : l'environnement au départ.
(b) : après la triangulation de Delaunay.
(c) : distance minimum entre les angles et les murs.
(d) : triangulation de Delaunay en prenant en compte les distances minimum

La triangulation de Delaunay

Elle permet d'obtenir une représentation exacte de l'environnement en 2D. Cette méthode consiste en la triangulation unique de l'environnement de telle sorte qu'un cercle passant par les trois points d'un triangle ne contienne aucun autre point (figure 1.9). F. Lamarche propose une version filtrée de cette triangulation permettant de choisir une augmentation ou une diminution du nombre de triangles [**1**].

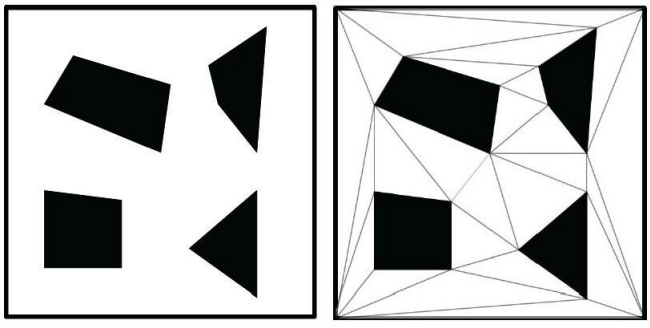
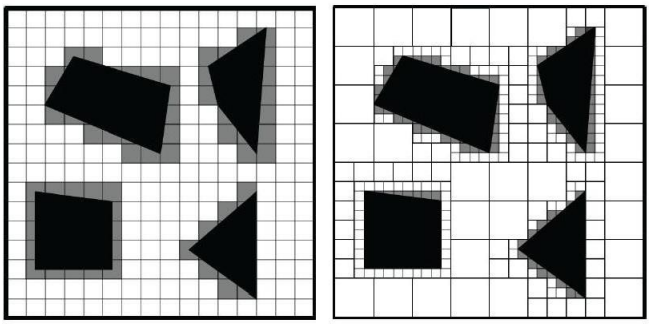


FIGURE 1.9 – triangulation de Delaunay

Grilles

Elle permet d'obtenir une représentation approximative de l'environnement en 2D ou en 3D (Figure 1.10). Cette méthode divise l'environnement en cellules plus ou moins grandes et les légende de trois façons : libre, partiellement obstruée ou obstacle. Les grilles peuvent aussi être hiérarchisées par une succession de grille de plus en plus précise, organisée en arbre [**1**].

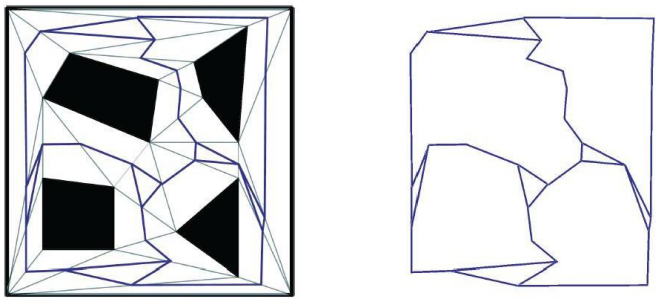


(a) Grille régulière (b) *Quadtree* de profondeur 3

FIGURE 1.10 – Grilles.

Cartes de cheminement

Cette méthode permet de calquer un environnement 3D en 2D, rendant ainsi les calculs moins longs car les déplacements y sont désormais plans (Figure 1.11). Les plans 2D ainsi extraits de l'environnement peuvent être légendés de façon à faire apparaître les bâtiments comme obstacles pour une navigation urbaine et de marquer les zones navigables. Ces zones peuvent également être divisées en les numérotant, permettant ainsi d'extraire un graphe spatial hiérarchique liant les zones entre elles. Le choix de navigation entre les zones peut donc s'effectuer en parcourant ce graphe. Les zones navigables peuvent également être représentées sous forme de réseaux de nœuds. Ces nœuds peuvent ainsi être pondérés en fonction de différents critères pour permettre à l'agent de choisir son chemin. Cette méthode se base sur une notion d'équidistance aux obstacles de l'environnement. Des sites sont évalués au sein de l'environnement, correspondant aux obstacles, dont les intersections vont former les points clefs. Les chemins générés maximisent la distance aux obstacles [**1**].

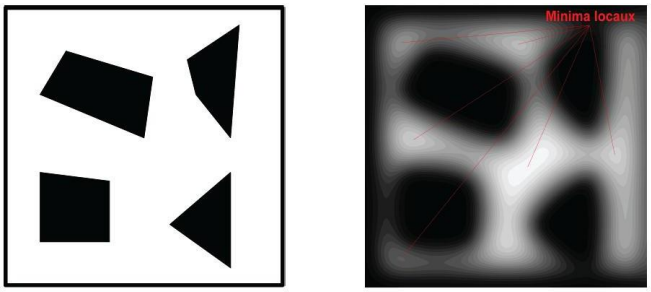


(a) Calculs sur l'environnement d'origine (b) Exemple de carte de cheminement

FIGURE 1.11 – Cartes de cheminement.

Champs de potentiel

Afin de résoudre directement le déplacement des personnes, une méthode utilisant des champs de potentiel peut également être appliquée à l'environnement de telle sorte que les obstacles aient une force de répulsion tandis que les buts une force d'attraction (Figure 1.12). Les notions de répulsion et d'attraction sont présentées plus en détail et illustrées dans la littérature [**1**].



(a) Environnement d'origine (b) Exemple de champs de potentiels

FIGURE 1.12 – champs de potentiel

Conclusion

Ces domaines d'applications variés, justifient les recherches effectuées en simulation comportementale, et particulièrement le besoin d'établir un lien avec les études effectuées sur le comportement humain. Les modèles proposés, pour permettre d'obtenir un réalisme suffisant, doivent généralement prendre en compte les caractéristiques propres au comportement humain.

Nous avons présenté au niveau de ce chapitre la modélisation des comportements des humains virtuels comme nous avons cité la plupart des modèles comportementaux utilisés. Dans le chapitre suivant, nous allons présenter un comportement de base dans la simulation comportementale: c'est le comportement de recherche de chemin multi-objectif.

CHAPITRE 2

RECHERCHE DE CHEMIN ET OPTIMISATION-MULTI-OBJECTIFS

Introduction

La recherche de chemin est un composant fondamental de nombreuses applications importantes dans les domaines du GPS, des jeux vidéo, de la robotique, de la simulation de foules et peut être implémenté de manière statique, dynamique et réelle. Bien qu'un certain nombre de développements aient amélioré la précision et l'efficacité des techniques de reconnaissance de trajectoire au cours des deux dernières décennies, le problème suscite encore de nombreuses recherches. Actuellement, le domaine le plus important concerne la fourniture de chemins réalistes et performants aux utilisateurs. En général, il existe différentes variantes du problème de recherche de cheminement, telles que la recherche de chemin par un agent unique, la recherche de chemin par plusieurs agents, la trajectoire contradictoire, les modifications dynamiques de l'environnement, un terrain hétérogène, des unités mobiles et des informations incomplètes. Chacun de ces problèmes a différentes applications dans différents domaines.

Recherche de chemin

La recherche de chemin consiste généralement en deux étapes principales: la génération de graphes et un algorithme de cheminement. Les stratégies de recherche de chemin sont généralement utilisées comme noyau de tout système de mouvement.

Définition

La recherche de chemin, couramment appelée pathfinding, est un problème de l'intelligence

artificielle qui se rattache plus généralement au domaine de la planification et de la recherche de solution. Il consiste à trouver comment se déplacer dans un environnement entre un point de départ et un point d'arrivée en prenant en compte différentes contraintes.

Notions de la théorie des graphes

L'histoire de la théorie des graphes débute peut-être avec les travaux d'Euler au XVIIIe siècle et trouve son origine dans l'étude de certains problèmes, tels que celui des ponts de Königsberg (voir page de couverture, les habitants de Königsberg se demandaient s'il était possible, en par- tant d'un quartier quelconque de la ville, de traverser tous les ponts sans passer deux fois par le même et de revenir à leur point de départ), la marche du cavalier sur l'échiquier ou le problème de coloriage de cartes. La théorie des graphes s'est alors développée dans diverses disciplines telles que la chimie, la biologie, les sciences sociales. Depuis le début du XXe siècle, elle constitue une branche à part entière des mathématiques, grâce aux travaux de König, Menger, Cayley puis de Berge et d'Erdös. De manière générale, un graphe permet de représenter la structure, les connexions d'un ensemble complexe en exprimant les relations entre ses éléments : réseau de communication, réseaux routiers, interaction de diverses espèces animales, circuits électriques. Les graphes constituent donc une méthode de pensée qui permet de modéliser une grande variété de problèmes en se ramenant à l'étude de sommets et d'arcs. Les derniers travaux en théorie des graphes sont souvent effectués par des informaticiens, du fait de l'importance qu'y revêt l'aspect algorithmique.

Définition

Théorie des graphes, branche des mathématiques concernée par les réseaux de points reliés par des lignes. Le sujet de la théorie des graphes a ses débuts dans les problèmes mathématiques récréatifs (voir le jeu des nombres), mais il est devenu un domaine important de la recherche mathématique, avec des applications en chimie, en recherche opérationnelle, en sciences sociales et en informatique.

Définition formelle

Un graphe G est un couple (V, E) où :

— V = (v1, v2, . . . , vn) Est l'ensemble fini dont les éléments sont appelés sommets ou nœuds.

— E = (e1, e2, . . . , em) Est l'ensemble fini du produit cartésien V. V dont les éléments sont appelés arcs ou arête.

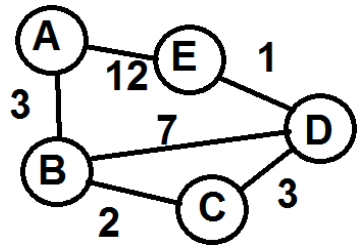


FIGURE 2.1 – Graphe.

Quelques conventions et notations

1. Graphe non orienté : est un graphe dans lequel les bords n'ont pas d'orientation. L'arête (a, b) est identique à l'arête (b, a), c'est-à-dire que ce ne sont pas des paires ordonnées, mais des ensembles *u, v* de sommets.

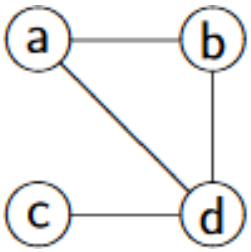


FIGURE 2.2 – Graphe non-orienté.

2. Graphes orientés : On dit qu'un graphe est orienté si les arêtes sont à sens unique. On les représente donc avec une flèche sur les dessins.
3. Graphe dirigé est un graphique dans lequel chaque arête est représentée par une paire ordonnée de deux sommets, par ex. (Vi, Vj) désigne une arête de Vi à Vj (du premier sommet au deuxième sommet).
4. Graphes pondérés : On définit un graphe pondéré comme un graphe où chaque arête et chaque sommet est affecté d'un nombre réel positif, appelé poids : Cela peut-être une

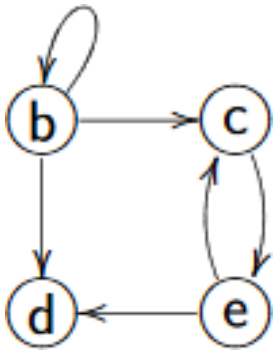


FIGURE 2.3 – Graphe orienté.

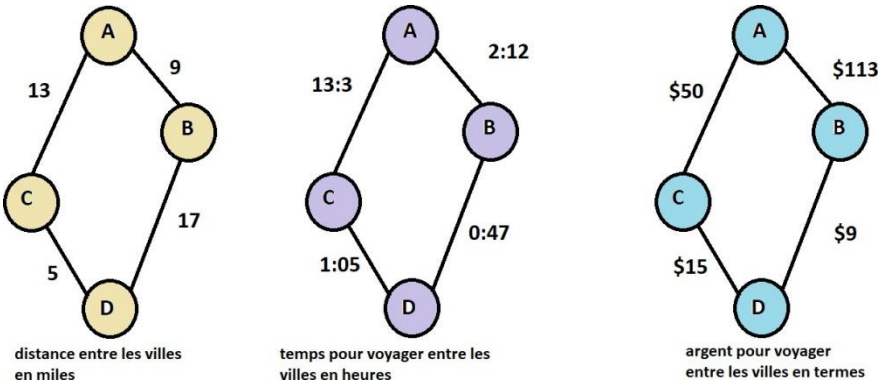


FIGURE 2.4 – Graphe pondéré.

distance : lorsque que l'on cherche un plus court chemin entre deux nœuds, il va de soit que la somme des pondérations des arêtes doit être aussi petit que possible. Mais ça peut aussi être un réel dans [0 ; 1] pour désigner des probabilités dans les chaînes de Markov par exemple. Cela peut aussi être un score, un prix, etc.

5. **Adjacents :** Sommets adjacents : dans un graphe G = (V, E), deux sommets sont dits adjacents (voisins), s'il existe une arête entre les deux sommets
6. **Connexité :** Un graphe est dit connexe lorsqu'il existe un chemin entre toute paire de nœuds. Une composante connexe d'un graphe est un sous-graphe connexe de ce graphe. Un sous-graphe est un sous-ensemble de nœuds du graphe, avec une partie de leurs arêtes associée.
7. **Degré d'un sommet et degré d'un graphe :**
   — **Degré d'un sommet :** On appelle degré du sommet v, et on note d(v), le nombre

d'arêtes incidentes à ce sommet.

— **Degré d'un graphe :** Le degré d'un graphe est le degré maximum de tous ses sommets.

Représentation graphique

L'agent doit effectuer une recherche de chemin sur un réseau spatial. Le réseau, appelé aussi une représentation graphique, est un composant de base de la recherche de chemin. Il y a plusieurs façons de présenter un graphe. Nous les classons en trois classes, qui sont points de cheminement, maillages de navigation et grilles. Dans les graphes de points de cheminement (Figure 2.5), chaque nœud est appelé un point de cheminement et spécifie un emplacement dans la région ; les points de cheminement sont reliés par des arcs différents. Un arc entre deux points de cheminement signifie que l'agent peut suivre ce chemin sans aucune collision. L'idée principale du graphe de point de cheminement est de mettre quelques points et arcs sur la carte pour que les agents puissent trouver leur chemin vers la destination autour d'obstacles statiques. Lorsque les obstacles et les points sont fixés sur une certaine carte, il n'y a pas de changement du chemin le plus court entre deux nœuds. De cette façon, quelques optimisations ont été faites en fonction du point depuis les points pourrait être prétraité sur la carte avant l'exécution réelle de la recherche de chemin. Cependant, l'inconvénient est que, lorsqu'il y a plusieurs personnages, les agents peuvent rester bloqués sur le point de cheminement. En revanche, lorsque la carte est fréquemment modifiée ou les obstacles sont mobiles, le système doit à nouveau prétraiter la carte, ce qui pourrait très cher.

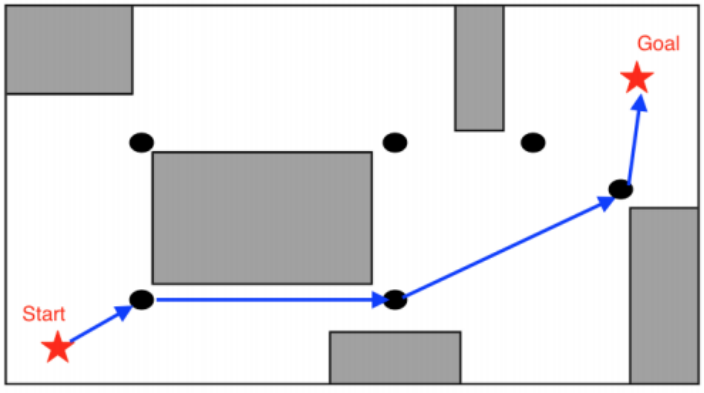


FIGURE 2.5 – Graphe des points de cheminement.

Le maillage de navigation (Figure 2.6), également appelé navmesh, est formé par un groupe de polygones convexes connectés, dans lequel chaque polygone définit une zone navigable de l'environnement (pas d'obstacles). À chaque polygone, l'agent peut accéder à n'importe quel point de cette zone, et à tout autre point de la même zone puisque le polygone est

convexe. Semblable au graphique de points de cheminement, le système doit prétraiter la carte pour dessiner le maillage de navigation afin d'éviter les obstacles, et l'agent peut atteindre l'objectif lorsqu'il parcourt une série de polygones.

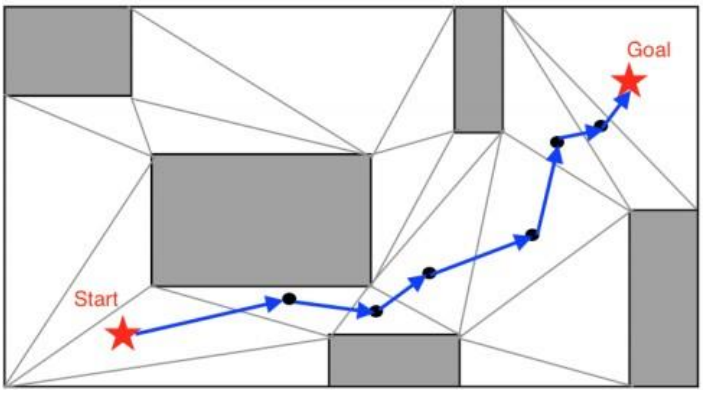


FIGURE 2.6 – Graphique de maillage de navigation

Le troisième est le quadrillage (Figure 2.7), qui est le plus utilisé en recherche. La recherche de chemin basée sur la grille est requise dans de nombreux jeux vidéo et mondes virtuels pour déplacer les agents. Les graphiques en grille sont constitués d'une collection de tuiles. Chaque tuile est considérée comme un nœud dans le graphe, et il peut être défini comme une tuile traversable ou non traversable (obstacles).

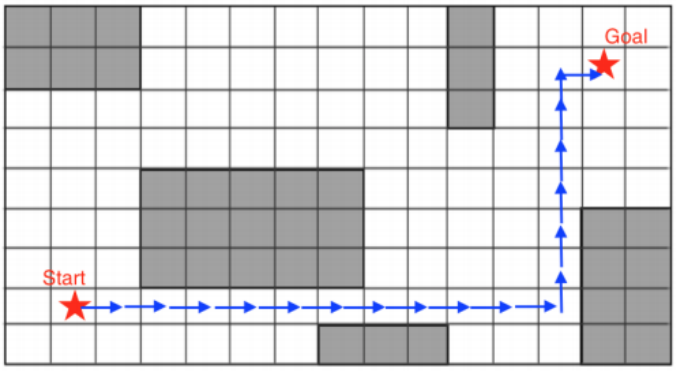


FIGURE 2.7 – Grille graphique

Les algorithmes de recherche de chemin

Les algorithmes de recherche de chemin permettent de trouver des itinéraires possibles du point source souhaité au point destination en utilisant les informations de localisation obtenues

numériquement. Ces algorithmes sont utilisés pour résoudre différents problèmes dans différents domaines. À titre d'illustration, certains des domaines les plus importants sont le développement de jeux, l'intelligence artificielle, les robots mobiles et les véhicules sans pilote.

Algorithme de Bellman Ford :

L'algorithme de Bellman Ford est utilisé pour trouver les chemins les plus courts du sommet source à tous les autres sommets dans un graphe pondéré. Cela dépend du concept suivant : le chemin le plus court contient la plupart des arcs, car le chemin le plus court ne peut pas avoir de cycle.

Algorithm 1 Pseudo-algorithme de Bellman-Ford

```
procedure BELLMAN-FORD(G, w, s)
    Pour tout sommet v ∈ V faire // Initialisation
    d[v] ←, ∞, π[v] ←, NIL
    d[s] ← 0
    Pour tout arc (u, v) ∈ E faire // relâchement de l'arc (u, v)
    Si d[v] > d[u] + w(u, v) alors
    d[v] ← d[u] + w(u, v), π[v] ← u
    Pour tout arc (u, v) ∈ faire // détection des circuits négatifs
    Si d[v] > d[u] + w(u, v) alors
    retourner Faux
end procedure
```

Algorithme de Floyd – Warshall :

L'algorithme de Floyd – Warshall est utilisé pour trouver les chemins les plus courts entre toutes les paires de sommets dans un graphe, où chaque arête du graphe a un poids qui est positif ou négatif. Le plus grand avantage de l'utilisation de cet algorithme est que toutes les distances les plus courtes entre tous les sommets peuvent être calculées en, où est le nombre de sommets dans un graphe.

Algorithm 2 Pseudo-algorithme Floyd-Warshall

1: Entrée : Un digraphe G avec V (G) = 1,. . ., n et poids c :
2: E (G) → R
3: Sortie : Une matrice $n \times$ n M telle que M [i, j] contienne la longueur d'un chemin le plus court du sommet *i* au sommet *j*.
4: M [i, j] : = ∞ ∀= j
5: 2 M [i, i] : = 0 $_i$∀
6: M [i, j] : = c ((i, j)) ∀(i, j) ∈(G)
7: Pour i : = 1 à n faire
8: Pour j : = 1 à n faire
9: Pour k : = 1 à n faire
10: Si M [j, k]> M [j, i] + M [i, k] alors M [j, k] : = M [j, i] + M [i, k]
11: Pour i : = 1 à n faire
12: Si M [i, i] <0 then
13: return ('graphe contient un cycle négatif')

Algorithme de Dijkstra :

L'idée derrière cet algorithme est la suivante : étant donné un ensemble de sommets sur un graphe et le coût de parcours de chaque nœud par rapport à notre position de départ, nous choisissons d'explorer les nœuds dont le coût est le minimum et élargir ses fils à partir de là [14]. Des comparaisons peuvent être faites entre le chemin actuel et des chemins potentiellement plus courts afin de trouver un vrai chemin optimal. En choisissant seulement de suivre les arêtes que nous estimons être les moins chères, nous pouvons éventuellement obtenir le chemin le plus court. L'algorithme de Dijkstra est un exemple d'algorithme de recherche basé sur le concept de navigation dans les graphes orientés tout en minimisant la «distance parcourue» en même temps. Le problème est de déterminer un chemin dans lequel le nombre de ressources dépensées peut être minimisé, car par rapport à des problèmes tels que le problème du voyageur de commerce référencé dans le travail original de Dijkstra. Si la distance parcourue peut être minimisée, alors naturellement le temps et les ressources dépensées pour atteindre une destination donnée seront également minimes. Dijkstra utilise l'idée que chacun des nœuds d'un graphe peut être décomposé en différents ensembles afin de définir leurs états comme suit :

1. Visité
2. Suivant à visiter
3. Pas encore visité

Le nœud de départ devient le nœud visité initial dans l'ensemble et de là l'algorithme se ramifie

aux nœuds voisins, en comparant plusieurs chemins du nœud actuel et en choisissant le chemin le plus court entre eux. Des chemins prolongeant les chemins nouvellement visités, le prochain le plus petit chemin vers les nœuds voisins peut être déterminé, en s'accumulant à partir du poids du chemin précédemment parcourus. Suivant cette idée de ne choisir que les chemins les plus courts en fonction du nœud en cours visité, le chemin minimal partant du nœud d'origine vers tous les autres nœuds (et donc un nœud choisi) peut être déterminé [4].

Algorithm 3 Pseudo-algorithme de Dijkstra

```
1: Entrée : démarrage du nœud, objectif du nœud
2: Principal ()
3: openList : = 0/ ; // file d'attente prioritaire basée sur g
4: ClosedList : = 0/ ;
5: g (début) : = 0 ;
6: openList.Insert (début) ;
7: Tant que openList = 0/
8: courant : = openList.Pop () ;
9: Si l'objectif actuel == alors
10: retourne 'chemin trouvé' ;
11: ClosedList.Insert (courant) ;
12:
13:  Pour chaque voisin ∈ courant
14: SI voisin dans ClosedList
15: continuer ;
16: Sinon voisin ouvert
17: openList.Insert (voisin) ;
18: Si g (courant) + c (courant, voisin) <g (voisin)
19: g (voisin) : = g (courant) + c (courant, voisin) ;
20: parent (voisin) : = courant ;
21: retourne «chemin non trouvé»;
22: fin ;
```

Algorithme A* :

En utilisant l'algorithme de Dijkstra comme base, l'algorithme A * tente d'obtenir un plus court chemin plutôt que de choisir avidement le voisin le plus proche sans aucune connaissance de l'endroit où se trouve le but. Pour que cet algorithme fonctionne, tous les nœuds du graphe sont nécessaires pour suivre leur propre coût secondaire qui représente la distance estimée par rapport à la position de l'objectif. Chaque nœud est conscient à la fois de son coût de traversée depuis le nœud de départ et de sa distance estimée du but. L'utilisation de ces valeurs clés (désignées par une fonction g (x) et h (x) respectivement), une recherche plus informée que celle de l'algorithme de Dijkstra peut avoir lieu et permet à l'agent d'étendre les nœuds qui sont beaucoup plus proches de son chemin préféré [4].

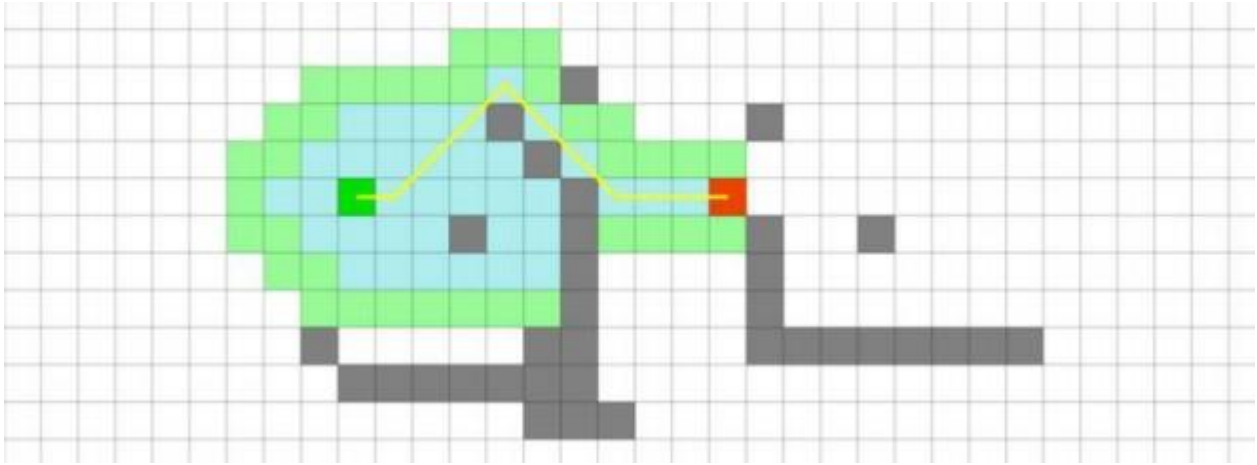


FIGURE 2.8 – Algorithme A Étoile.

À un niveau élevé, A * peut être utilisé pour trouver le chemin le plus court dans un graphe dirigé à partir d'un point de départ s et un point final g. L'algorithme utilise les connaissances basées sur les nœuds environnants à partir du point actuel et calcule l'itinéraire le plus court possible jusqu'au nœud suivant, tout en tenant compte de la distance réelle entre le nœud actuel et le nœud de destination. La fonction est décrite comme :

$$f(n) = g(n) + h(n)$$

où g (n) calcule la distance entre le nœud actuel et le nœud suivant, et h (n) calcule la distance estimée du nœud actuel au nœud de destination. La distance minimale retournée à partir de cette fonction sur une série de nœuds se trouve l'itinéraire minimisé et donc le chemin le plus court. Il suit une idéologie de recherche étendue en premier, en analysant la liste ouverte, qui est une liste de points disponibles pour traverser à partir du nœud actuel. La valeur g (n) est calculée comme le coût pour passer d'un nœud à la suivante, alors que la valeur h (n) représente l'heuristique ou la distance réelle d'un nœud donné au but (en ignorant les obstacles et les collisions). Tout nœud placé dans la liste fermée devient le nœud suivant pour lequel recalculer g (n) et le cycle continue jusqu'à ce que le nœud de but soit atteint, à quel point un chemin peut être tracé du but vers le nœud de départ.

Optimisation multi objectif

Les ingénieurs se heurtent quotidiennement à des problèmes technologiques de complexité grandissante, qui surgissent dans des secteurs très divers, comme dans le traitement des images,

Algorithm 4 Pseudo-algorithme A *

```
1: Entrée : démarrage du nœud, objectif du nœud
2: Principal ()
3: openList : = 0/ // file d'attente prioritaire basée sur f
4: ClosedList : = 0/
5: g (début) : = 0 ;
6: f (début) : = g (début) + h (début) ;
7: openList.Insert (début) ;
8: Tant que openList == 0/
9: courant : = openList.Pop () ;
10: Si l'objectif actuel == alors
11: retourne «chemin trouvé»;
12: ClosedList.Insert (courant) ;
13:
14: Pour chaque voisin ∈ courant.
15: Si voisin dans ClosedList alors
16: continuer ;
17: Sinon voisin ouvert alors
18: openList.Insert (voisin) ;
19: Si g (courant) + c (courant, voisin) <g (voisin)
20: g (voisin) : = g (courant) + c (courant, voisin) ;
21: f (voisin) : = g (voisin) + h (voisin) ;
22: parent (voisin) : = courant ;
23: retourne «chemin non trouvé»;
24: fin ;
```

la conception des systèmes mécaniques, la recherche opérationnelle, ... Le problème à résoudre peut fréquemment être exprimé sous la forme générale d'un problème d'optimisation, dans lequel on définit une fonction objective, ou fonction de coût (voire plusieurs), que l'on cherche à minimiser par rapport à tous les paramètres concernés. Par exemple, dans le célèbre problème du voyageur de commerce, on cherche à minimiser la longueur de la tournée d'un « voyageur de commerce », qui doit visiter un certain nombre de villes, avant de retourner à la ville de départ. La définition du problème d'optimisation est souvent complétée par la donnée de contraintes : tous les paramètres (ou variables de décision) de la solution proposée doivent respecter ces contraintes, faute de quoi la solution n'est pas réalisable [5].

La valeur optimale ou la meilleure solution peut être trouvée grâce au processus d'optimisation. Les problèmes d'optimisation incluent la recherche d'une valeur maximale ou minimale ou l'utilisation d'un objectif ou multi-objectif. Les problèmes qui ont plus d'un objectif sont appelés optimisation multi-objectifs (MOO). Ce type de problème se retrouve dans la vie de tous les jours, comme les mathématiques, l'ingénierie, les études sociales, l'économie, l'agriculture, l'aviation, l'automobile et bien d'autres [6].

L'optimisation multio bjectif est un axe de recherche très important à cause de la nature multi objectif de la plupart des problèmes réels. Les premiers travaux menés sur les problèmes multi objectifs furent réalisés au 19éme siècle sur des études en économie par Edgeworth et généralisés par Pareto.

Dans la plupart des problèmes du monde réel, il ne s'agit pas d'optimiser seulement un seul critère mais plutôt d'optimiser simultanément plusieurs critères et qui sont généralement conflictuels. Dans les problèmes de conception, par exemple, il faut le plus souvent trouver un compromis entre des besoins technologiques et des objectifs de coût. L'optimisation multi objective consiste donc à optimiser simultanément plusieurs fonctions.

Un problème :

Un problème dans le point de vue informatique veut dire comment faire relier un ensemble de données par un ensemble de résultats, où les données vont subir un ensemble d'opérations dont on les appelle le traitement. Donc il est conçu comme une relation $H \pm A$ x S entre les entrées ou instances, et les sorties ou solutions.
Pour qu'une instance d'un problème soit compréhensible par un ordinateur numérique, elle doit être décrite comme une séquence finie de symboles d'un ensemble fini arbitraire appelé alphabet. De même, la solution d'une instance d'un tel problème est émise dans ce format.

Généralement, l est infini alors que S peut être fini.

Types des problèmes :

Les problèmes peuvent être classés selon les propriétés de l'ensemble des solutions :

- **Un problème de décision :** c'est un problème dont la réponse est tout simplement OUI ou NON. [5]
- **Un problème polynomial réductible :** on a L1 et L2 deux problèmes de décision, on dit L1 est polynomial réductible à L2 s'il existe un algorithme polynomial qui convertit chaque instance d'entrée de L1 à une autre instance de L2 [5].
- **Un problème de la classe P :** un problème est dit polynomial s'il existe un algorithme de complexité polynomiale permettant de répondre à la question posée dans ce problème, quelque soit la donnée de celui-ci. La classe P est l'ensemble de tous les problèmes de reconnaissances polynomiaux [5].
- **Un problème de la classe NP :** un problème de reconnaissance est dans la classe NP si, pour toute instance de ce problème, on peut vérifier, en un temps polynomial par rapport à la taille de l'instance, qu'une solution proposée ou devinée permet d'affirmer que la présence est « oui » pour cette instance [5].
- **Un problème de la classe NP-hard :** on dit qu'un problème est dans la classe NP-hard si chaque autre problème dans NP est réductible d'une manière polynomiale dans ce dernier.
- **Un problème de la classe NP-complet :** s'il appartient à NP et chaque autre problème dans NP est réductible d'une manière polynomiale dans ce dernier. C'est un problème de décision et NP-hard qui cherche à trouver l'optimum [5].

Fonction de coût (objective) :

C'est le nom donné à la fonction f (on l'appelle encore fonction de coût ou critère d'optimisation). C'est cette fonction que l'algorithme d'optimisation va devoir « optimiser » (trouver un optimum).

Mathématiquement parlant, un problème d'optimisation se présentera sous la forme suivante :

$$\min \quad f(\vec{x}) \qquad \text{(fonction à optimiser)}$$
$$\text{avec} \quad \vec{g} = (\vec{x}) \qquad \text{(m contraintes d'inégalité)}$$
$$\text{et} \quad \vec{h} = (\vec{x}) \qquad \text{(p contraintes d'égalité)}$$

On a $\vec{x} \in \Re^n$, $\vec{g}(\vec{x}) \in \Re^m$ et $\vec{h}(\vec{x}) \in \Re^p$

Ici les vecteurs $\vec{g}(\vec{x})$ et $\vec{h}(\vec{x})$ représentent respectivement **m** contraintes d'inégalité et **p** contraintes d'égalité [5].

Un problème d'optimisation multi objectif :

La plupart des problèmes d'optimisation sont décrits par plusieurs objectifs ou critères souvent contradictoire qui doivent être optimisés simultanément. Ils produisent un problème multi objectifs, dès lors résoudre un problème multi objectifs ne consiste pas à chercher la solution optimale, on cherche à avoir une solution tout en respectant la présence d'un ensemble de contraintes. La solution c'est un résultat de faire combiner ces contraintes ensemble d'une manière qu'on maximise quelques uns et on minimise les autres, ces contraintes ont une caractéristique primordiale, c'est que chaque contrainte influe sur les autres soit quand on minimise sa valeur ou on la maximise, dans un autre terme on dit que les contraintes sont conflictuelles [5]. La difficulté principale d'un problème d'optimisation multi objectifs est qu'il n'existe pas de définition de la solution optimale.

Par exemple, Prenons le cas d'une personne souhaitant acheter téléphone portable de bonne qualite et plusieurs options. Ce téléphone mobile est idéal quand il est de haute qualité et a plusieurs bonnes caractéristiques. Notre acheteur va donc devoir identifier les meilleurs compromis possibles correspondant à son budget.

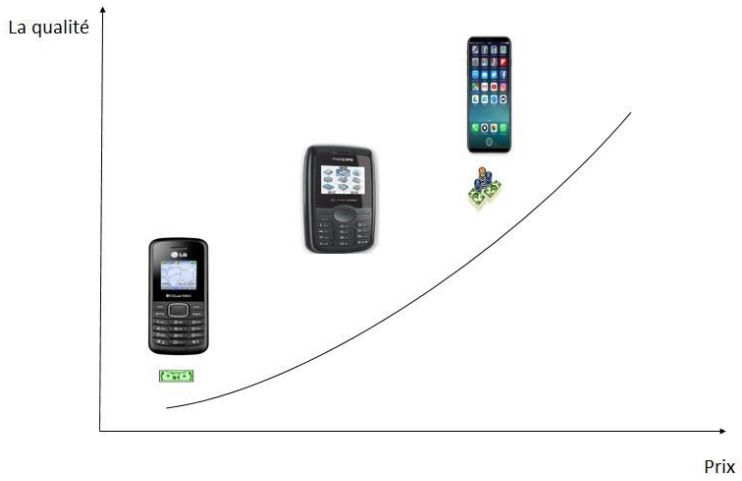


FIGURE 2.9 – Un problème combinatoire.

Formulation d'optimisation multi objectifs

Un **problème multi objectifs** ou **multicritère** peut être défini comme un problème dont on recherche l'action qui satisfait un ensemble de contraintes et optimise un vecteur de fonctions objectifs [7].

Définition :

La définition mathématique d'un problème d'optimisation multi objectifs.

Une action (ou **un vecteur de décision**) sera note :

$$x = (x_1, x_2, x_3, \ldots, x_n) \ (1)$$

Les contraintes seront notées :

$$g_i(x) \ avec \ i = 1 \ldots m \ (2)$$

avec m le nombre de contraintes

Le vecteur de fonction objectifs sera notée f :

$$f(x) = (f_1(x), f_2(x), \ldots, f_k(x)) \ (3)$$

avec f_i objectifs ou critères de décision et k le nombre d'objectifs

Nous considérons sont les fonctions de minimisation.

Un problème d'optimisation recherche a l'action x^* telle que les contraintes $g_i(x^*)$ soient satisfaites pour i = 1, ...,m et que optimise la function $f(x^*) = (f_1(x^*), f_2(x^*), \ldots, f_k(x^*))$ [7]

L'ensemble E est résultat de l'union des domaine de définition de chaque variable et contraintes définies en (3) qu'on a appelle **l'ensemble des actions réalisables**.

On nomme **l'ensemble des objectifs realisable** par F.

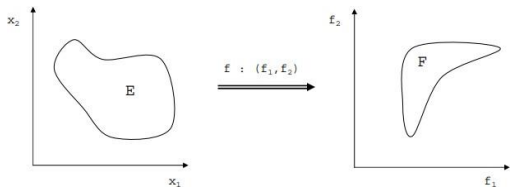


FIGURE 2.10 – Definition E,F,f
[7]

Les méthodes d'optimisation Multiobjectifs

Un grand nombre d'approches existent pour résoudre les problèmes multi objectifs. Certaines utilisent des connaissances du problème pour fixer des préférences sur les critères et ainsi contourner l'aspect multicritère du problème. D'autres mettent tous les critères au même niveau d'importance, mais l'a aussi il existe plusieurs façons de réaliser une telle opération. Plusieurs ouvrages ou articles de synthèse ont été rédigés. Parmi toutes ces approches, il faut distinguer deux catégories : les approches non Pareto et les approches Pareto. Les approches non Pareto ne traitent pas le problème comme un véritable problème multi objectif. Elles cherchent à ramener le problème initial à un ou plusieurs problèmes mono-objectifs. Par opposition aux méthodes non Pareto.

Nous allons dans cette section, d'écrire les principales approches non Pareto et commenter leurs avantages et leurs inconvénients [8].

Les méthodes d'agrégation des objectifs :

La première approche discuter ici est aussi la plus évidente. Elle consiste à transformer un problème multiobjectif en un problème à un objectif en agrégeant les différents critères sous la forme d'une somme pondérée[8] :

Minimiser $f_\Sigma = \sum_{i=1}^{m} w_i f_i(\vec{x})$

tel que $\vec{g}(\vec{x}) \leq 0$

avec $\vec{x} \in \mathbb{R}^n$, $\vec{f}(\vec{x}) \in \mathbb{R}^m$, $\vec{g}(\vec{x}) \in \mathbb{R}^q$

les w_i appelés poids, peuvent être normalisés sans perte de généralité : $\Sigma\ w_i = 1$. Il est clair que la résolution d'un problème pour un vecteur de poids $\vec{w}$ fixé ne permet de calculer quelques solutions Pareto optimales. Pour obtenir un ensemble contenant un grand nombre de solutions Pareto optimales, il faut résoudre plusieurs fois le problème en changeant à chaque fois les valeurs de $\vec{w}$ et p. Les deux points Pareto optimaux trouvés sont A et C. En faisant varier le vecteur

$\vec{w}$, il est possible de trouver d'autres points Pareto optimaux. Seulement, tous ces points se trouveront sur les parties convexes de la surface de compromis. Il n'existe pas de valeur possible pour $\vec{w}$ permettant de trouver par exemple le point B. En effet, cette approche ne permet pas d'approcher la totalité du front Pareto lorsque celui-ci est non convex.

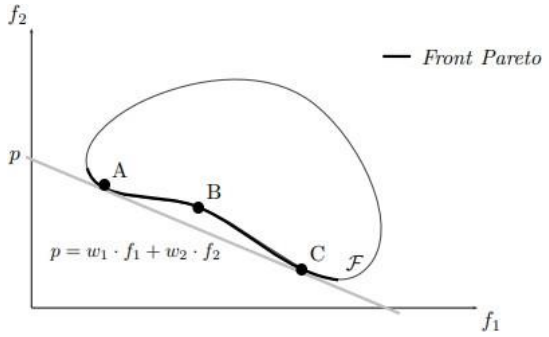


FIGURE 2.11 – Interprétation graphique de l'approche par pondération.

L'approche par contraintes

Une autre façon de transformer un problème d'optimisation multi objectif en un problème simple objectif est de convertir m - 1 des m objectifs du problème en contraintes et d'optimiser séparément l'objectif restant. Le problème peut être reformulé de la manière suivant [8] :

Minimiser $f_i(\vec{x})$

tel que $f_1(\vec{x}) \leq \varepsilon_1$

.

$f_{i-1}(\vec{x}) \leq \varepsilon_{i-1}$

$f_{i+1}(\vec{x}) \leq \varepsilon_{i+1}$

.

$f_{i+m}(\vec{x}) \leq \varepsilon_{i+m}$

tel que $\vec{g}(\vec{x}) \leq 0$

avec $\vec{x} \in \mathbb{R}^n$, $\vec{f}(\vec{x}) \in \mathbb{R}^m$, $\vec{g}(\vec{x}) \in \mathbb{R}^q$

L'approche par ε-contrainte doit aussi être appliquée plusieurs fois en faisant varier le vecteur $\vec{\varepsilon}$ pour trouver un ensemble de points Pareto optimaux.

Approche procédé. En transformant des fonctions objectives en contraintes, elle diminue la zone réalisable par paliers. Ensuite, le processus d'optimisation trouve le point optimal sur l'objectif restant.

L'approche Min-Max :

Cette approche consiste à transformer le problème multi objectifs en un problème à un seul objectif où l'on cherche à minimiser l'écart relatif par rapport à un point de référence appelé le but, fixé par la méthode ou décideur, Il existe plusieurs manières de caractériser la distance

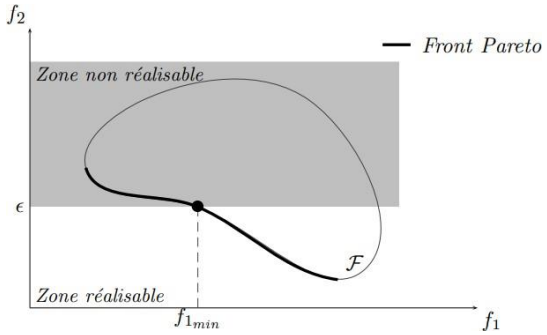


FIGURE 2.12 – Interprétation graphique de l'approche par ε -contrainte

entre un point de référence (le but) et un autre, notamment à l'aide de normes. Une norme est définie de la manière suivante [8] :

$$L_r(\vec{f}(\vec{x})) = [\sum_{i=1}^{m} |B_i - f_i(\vec{x})|]^{\frac{1}{r}}$$

Les principales normes utilisées sont : $L_1 = \sum_i = 1^m |B_i - f_i(\vec{x}|$, la distance classique et la norme $L_\infty = max_{i\in\{1,\ldots,m\}} |B_i - f_i(\vec{x})|$. C'est cette dernière qui est utilisée dans l'approche min-max appelée aussi approche de Tchebychev.

Minimiser	$max_{iin1,\ldots,m}(B_i - f_i(\vec{x}))$
tel que	$\vec{g}(\vec{x}) \leq 0$
avec	$\vec{x} \in \mathbb{R}^n$, $\vec{f}(\vec{x}) \in \mathbb{R}^m$, $\vec{g}(\vec{x}) \in \mathbb{R}^q$

Dans cette approche, le point de référence joue un rôle fondamental. S'il est mal choisis, la recherche peut s'avérer être très laborieux,

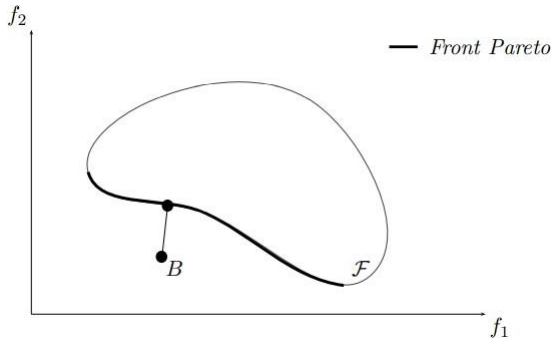


FIGURE 2.13 – Interprétation graphique de l'approche Mini-Max

Cas d'une recherche avec un but B fixé. Il est clair que l'approche permet de traiter les problèmes non convexes à condition que le point de référence soit choisi judicieusement. Les méthodes de résolution implémentant cette approche utilisant souvent le point idéal comme

point de référence. Ce point idéal évolue donc en fonction de la recherche. En effet, plus la surface de compromis courante e trouvée par la méthode se rapproche du front Pareto optimal, plus le point idéal se rapproche du point idéal du problème.

Le but à atteindre :

Cette approche, comme celle de min-max, utilise un point de référence pour guider la recherche. Mais elle introduit aussi une direction de recherche, si bien que le processus de résolution devra suivre cette direction. À la différence de l'approche min-max, qui utilise des normes pour formaliser la distance au point de référence, l'approche du but à atteindre utilise des contraintes, à l'instar de l'approche ε-contrainte, pour déterminer la position du point de référence (aussi appelé le but). L'écart par rapport à ce but est contrôle grâce à la variable λ introduite à cet effet [8].

Minimiser λ

tel que $f_1(\vec{x}) - w_1.\lambda \leq B_1$

.

$f_m(\vec{x}) - w_m.\lambda \leq B_m$

et que $\vec{g}(\vec{x}) \leq 0$

avec $\vec{x} \in \mathbb{R}^n$, $\vec{f}(\vec{x}) \in \mathbb{R}^m$, $\vec{g}(\vec{x}) \in \mathbb{R}^q$

Ainsi en minimisant λ et en vérifiant toutes les contraintes, la recherche va s'oriente vers le but B et s'arrêter sur le point A faisant partie de la surface de compromis. Cette approche permet, comme l'approche par ε-contrainte et l'approche min-max, de trouver les parties non convexes des fronts Pareto. Cependant, cette approche, comme les précédents, doit être itérée

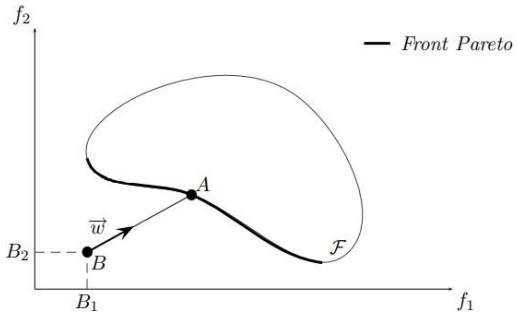


FIGURE 2.14 – Interprétation graphique de l'approche par "but à atteindre"

plusieurs fois dans le but d'obtenir un ensemble de points Pareto optimaux. Les paramètres $\vec{w}$ et B doivent être bien choisis par l'utilisateur. Bien que ses paramètres permettent une grande flexibilité de la recherche (orientation et but), s'ils sont mal choisis, ils peuvent, dans certains cas extrêmes, donner des résultats non cohérents.

Conclusion

Dans ce chapitre, nous avons présenté un état de l'art sur la recherche de chemin. On a également présenté quelques algorithmes de recherche de chemin populaires et la structure de données qui sont utilisés par ces algorithmes qui sont eux même des graphes , en prenant quelques notions de base sur les graphes, en suite on a passé à l'optimisation multi-objectifs Nous avons défini et nous avons mentionné quelques méthodes de solutions. Dans le chapitre suivant, nous montrerons la conception de notre solution.

CHAPITRE 3

CONCEPTION DU SYSTÈME

Introduction

La recherche de chemin est un composant fondamental de nombreuses applications importantes dans les domaines du GPS, des jeux vidéo, de la robotique, de l'animation comportementale (recherche de chemin (navigation), et la simulation de foules).

La recherche de chemin est un problème très bien étudié dans le domaine de l'informatique et il a de nombreuses applications dans le monde réel, telles que la détermination de l'itinéraire réseau le plus court, la navigation robotisée autonome, la détection du chemin le plus court entre la source et la destination sur une carte, etc.

Problématique

Notre problématique s'intéresse à la façon dont l'humain virtuel calcule le plus court chemin. Cela ne dépend pas seulement de la distance entre son point départ et la destination qu'il veut atteindre lors du calcul. Au lieu de cela, nous simulerons les autres critères on prenons en considération dans le comportement humain en choisissant le meilleur chemin.

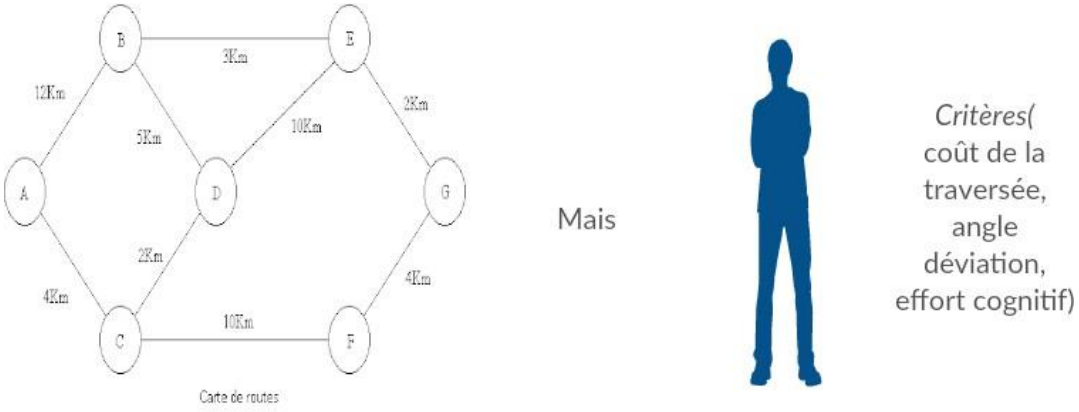


FIGURE 3.1 – Problématique.

Architecture du système

Conception globale

Le schéma de la figure 3.2 représente le système que nous avons proposé pour répondre à notre problématique. Ce schéma décrit la conception globale de notre système et comporte trois unités principales. Dans la première unité, le modèle d'environnement qui représente la structure de l'environnement. La deuxième unité ***agent*** qui explique les critères qu'un humain virtuel adopte pour choisir le meilleur chemin pour lui.

Conception détaillée

1 Modèle d'environnement

La planification de chemin pour déplacer des personnages dans des environnements virtuels est une tâche fondamentale dans les simulations. Les applications modernes plus en plus importante ; chaque personnage doit calculer et suivre un chemin de manière autonome tout en évitant des collisions avec des obstacles et d'autres personnages. Cela conduit à de nombreuses requêtes liées à la planification de chemin, suivi de chemin. Dans cette section, j'explique comment construire l'environnement dans lequel se déplace l'agent, c'est-à-dire identifier les obstacles statiques et dynamiques et comment les identifier et les éviter par l'agent.

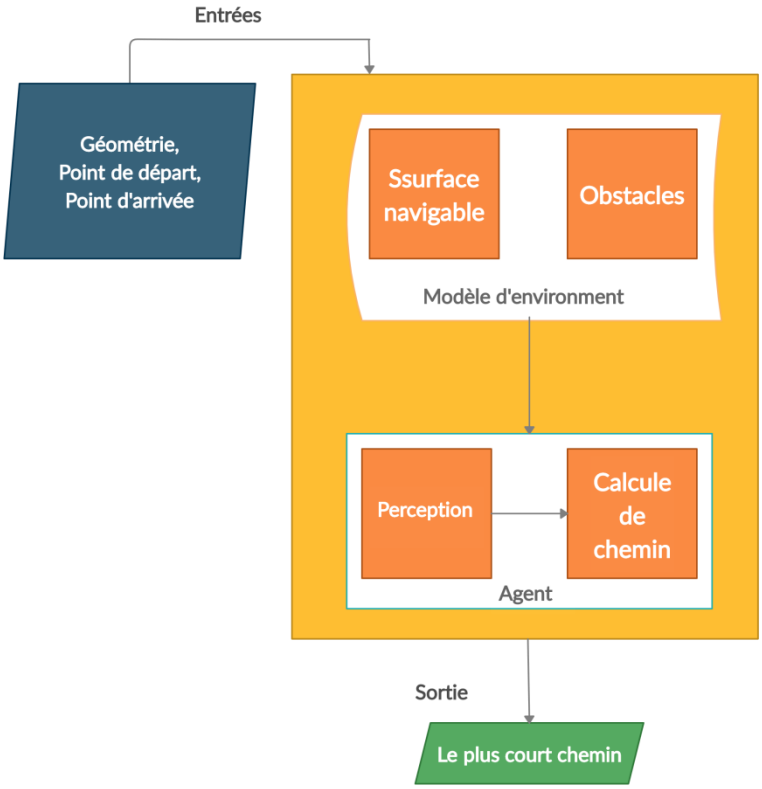


FIGURE 3.2 – Conception globale du système.

Surface navigable

Afin de naviguer au sein des environnements virtuels, les agents doivent pouvoir planifier leur chemin afin d'optimiser leurs déplacements. Il existe plusieurs modèles permettant de représenter l'espace libre de l'environnement afin d'y naviguer. L'espace libre, ces décompositions peuvent êtres exactes ou approximatives. Dans notre système on choisis la décomposition exacte, qui s'appelle maillage de navigation (Navigation Mesh).

Le maillage de navigation stocke cette surface sous forme de polygones convexes. Les polygones convexes sont une représentation utile, car nous savons qu'il n'y a aucune obstruction entre deux points à l'intérieur d'un polygone. En plus des limites des polygones, nous stockons des informations sur les polygones voisins les uns des autres. Cela nous permet de raisonner sur l'ensemble de la zone piétonnière.

Maintenant que nous avons une définition de l'espace libre, nous pouvons définir un maillage de navigation sous forme de tuple M = (R, G) : R = R0, R1,. . ..

- R est une collection de régions géométriques qui représente l'espace libre.
- G = (V, E) est un graphe non orienté qui décrit comment les caractères peuvent naviguer entre les régions de R. La Figure 3.3 montre un exemple abstrait de maillage de navigation.

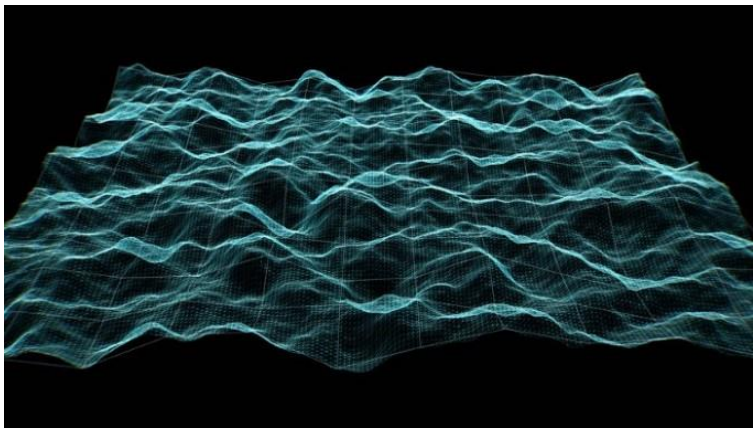


FIGURE 3.3 – Maillage de navigation

L'utilisation de cette structure de l'environnement dédiée permet un calcul rapide du chemin. L'environnement est ramené de trois à deux dimensions tout en conservant les informations utiles comme la hauteur des obstacles. Le problème se résout ensuite avec des algorithmes de parcours de graphes. Cette représentation qui utilise un graphe exprimant le voisinage entre les nœuds. Pour le calcul du chemin optimal on retrouve l'algorithme A-Étoile qui calcule le chemin optimal entre deux sommets en utilisant l'heuristique basé sur le cout de parcours (distance).

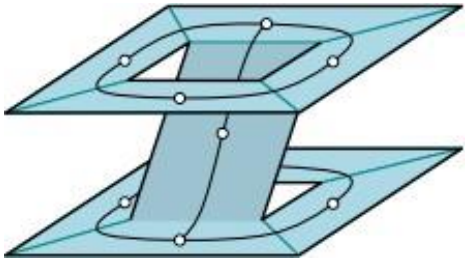


FIGURE 3.4 – Représentation abstraite de maillage de navigation.

Obstacles

Un obstacle peut avoir un impact sur la navigation soit sur au niveau mondial ou local. Pour le premier, il peut changer le maillage de navigation en y creusant des trous où ils se touchent. De ce fait, il devient une partie du maillage de navigation et est considéré au cours du processus de recherche de chemin.

Nous avons deux types d'obstacles que l'agent doit éviter lorsqu'il se rend à sa destination : obstacles statiques, obstacles dynamique. Ces obstacles influencent grandement sur la façon dont un agent détermine son chemin.

Obstacles statiques

Lorsque l'obstacle devient stationnaire, et peut être considéré comme bloquant le chemin de tous les agents, les obstacles devraient affecter la navigation globale, c'est-à-dire le maillage de navigation. Les obstacles statiques sont des structures. Cela affecte la façon dont un agent détermine son chemin tout en se rendant à sa destination.

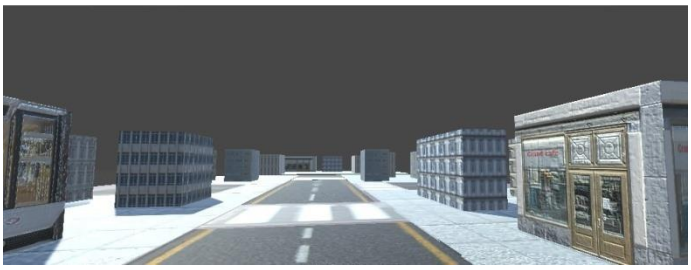


FIGURE 3.5 – Obstacles statiques

Obstacles Dynamiques

Lorsqu'un obstacle se déplace, il est préférable de le gérer en évitant les obstacles locaux. De cette façon, l'agent peut éviter l'obstacle de manière prédictive.
Les obstacles dynamiques sont des personnes et des voitures qui traversent l'environnement

dans des directions aléatoires. Les obstacles statiques sont des structures. Cela affecte la façon dont un agent détermine son chemin tout en se rendant à sa destination.

FIGURE 3.6 – Obstacles Dynamiques

2 Agent

Un agent peut être une entité physique ou virtuelle qui peut agir, percevoir son environnement (de manière partielle) et communiquer avec les autres, est autonome et possède les compétences pour atteindre ses objectifs et ses tendances. C'est dans ce système qui contient un environnement, des objets et des agents (les agents étant les seuls à agir), des relations entre toutes les entités, un ensemble d'opérations pouvant être effectuées par les entités et les changements de l'univers dans le temps et en raison de ces actions.

Perception

C'est une capacité qui lui permet d'acquérir des informations sur son environnement et sur lui-même . Elle est définie comme étant la prise de conscience de l'environnement grâce à des éléments de sensation physique.

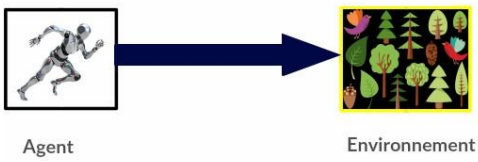


FIGURE 3.7 – Interaction de l'agent avec l'environnement.

L'agent accède directement à l'environnement pour faire la lecture des donnes du graphe avec lequel il fait le calcule du plus court chemin.

Calcule de chemin

Il existe plusieurs algorithmes de recherche de chemin le plus court. Dans notre projet on utilise A * pour calculer le chemin le plus court sur le maillage de navigation. A * fonctionne sur un graphe de nœuds connectés. L'algorithme part du nœud le plus proche du début du chemin et visite les nœuds de connexion jusqu'à ce que la destination soit atteinte.
Étant donné que la représentation de surface navigable est un maillage de polygones, la première chose que le pathfinder doit faire est de placer un point sur chaque polygone, qui est l'emplacement du nœud. Le chemin le plus court est alors calculé entre ces nœuds.
Les points et les lignes jaunes dans la figure 3.8 montrent comment les nœuds et les liens sont placés sur le maillage de navigation, et dans quel ordre ils sont traversés pendant le A *. Dans ce module, nous présenterons le processus de calcul du coût du chemin, qui est calculé à l'aide de l'algorithme A-étoile, et expliquerons les critères pris en considération lors du calcul du chemin le plus court.

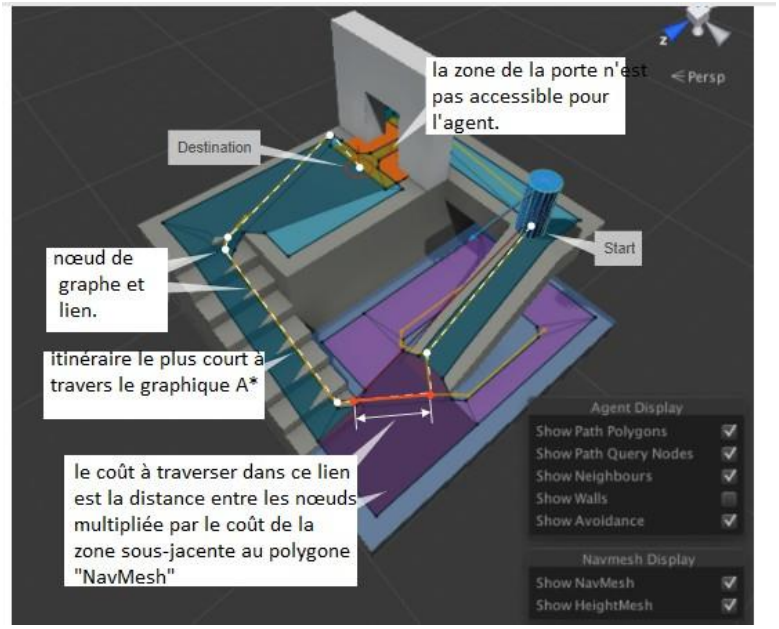


FIGURE 3.8 – Processus de calcule le chemin

Lors du calcul du chemin le plus court vers la destination, plusieurs critères sont pris en compte. Il y a ceux qui estiment la distance et ils y a ceux qui l'estiment par la distance eclidienne, et il y a ceux qui l'estiment par l'angle de déviation, et il y a ceux qui l'estiment par le nombre des tours entre le point de départ et le point d'arrivée.

Nous calculons ces valeurs à l'aide de la formule suivante :

— **F** est le coût total du nœud
— **G** est la distance entre le nœud actuel et le nœud de départ.
— **D** est la distance
— **C** Coût de zone navigable

$$f(n) = g(n) + d(n)*c(n)$$

Le coût de déplacement entre deux nœuds dépend de la distance à parcourir et du coût associé au type de zone du polygone sous le lien, c'est-à-dire distance * coût. En pratique, cela signifie que si le coût d'une zone est de 2,0, la distance à travers ce polygone apparaîtra deux fois plus longue. L'algorithme A * exige que tous les coûts soient supérieurs à 1,0.

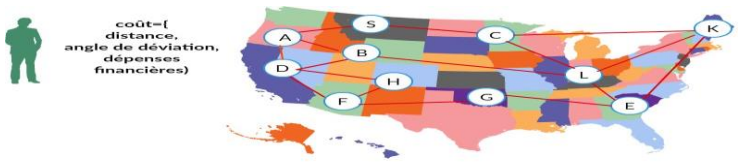


FIGURE 3.9 – Représentation sous forme de graphe.

Coût de la traversée

Coût de la traversée de chaque zones sont des subdivisions de la surface navigable. chaque subdivision a un coût entre dans l'opération de calcule du chemin, la figure (3.9) représente la subdivision de la surface navigable en des zones à différents coûts.

Par exemple comme nous avons vue dans la figure précédente chaque couleur représente le coût de la zone. En un mot, le coût nous permet de contrôler les zones privilégiées par l'explorateur lors de la recherche d'un chemin. Par exemple, si on définis le coût d'une zone sur 3,0, la traversée de cette zone est considérée comme trois fois plus longue que les itinéraires alternatifs.

Distance euclidienne

La distance euclidienne est la distance en ligne droite entre deux points, ce qui signifie que lorsqu'il n'y a pas d'obstacle sur le chemin, l'agent peut se déplacer directement du point

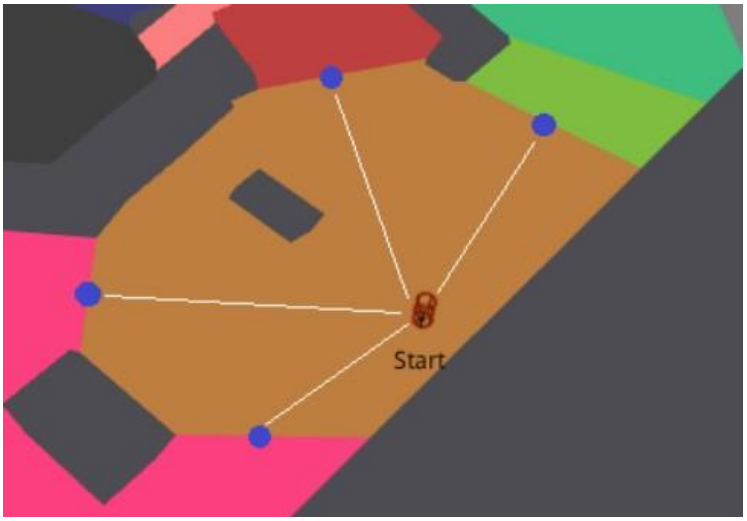


FIGURE 3.10 – représentation de graphe

de départ au point final sans aucun virage. De cette façon, la longueur du chemin est plus courte que la distance de Manhattan et la distance d'octile (théorème d'inégalité de triangle). La formule suivante peut déterminer la distance euclidienne

$$d(p1, p2)\_eclidienne = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

La formule de calcule le chemin devient être :

f(n) = g(n) + d(distace eclidienne)*c(cout de zone)

Théorème 1 *Théorème de Pythagore*

$D^2 = (Cot\ oppose)^2 + (Cot\ adjacent)^2$

La figure (3.10) représente le calcule de la distance par le théorème de Pythagore.

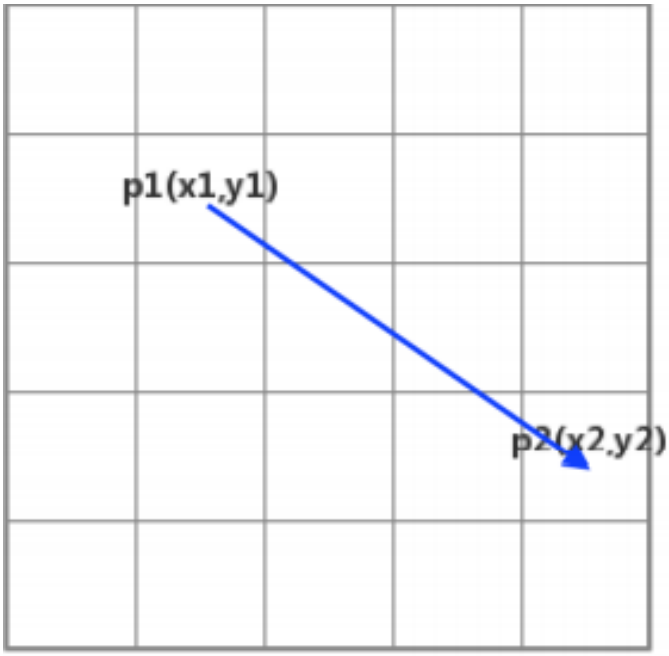


FIGURE 3.11 – Distance euclidienne.

Angle de déviation

Dans ce cas, l'angle de déviation minimum entre le lieu de départ et la destination est considéré comme la distance de chemin la plus courte. Il y a des personnes lors du choix du chemin le plus court, ils vont prendre en compte l'angle de déviation minimal lors du calcul du plus court chemin pour atteindre leur destination.

Dans ce cas l'agent calcul l'angle de déviation entre le vecteur menant directement à la destination et les vecteurs conduisant à tous les nœuds adjacents au nœud dans lequel se trouve l'agent. Puis il choisit le plus petit angle.

L'agent recherche le chemin le plus court en suivant le moindre angle de déviation entre lui et la destination qu'il souhaite atteindre. L'agent choisit le plus petit écart entre lui et la destination. La figure 3.12 le chemin dans le cas le cout calculé selon l'angle de déviation. Donc la fonction de cout :

$$f(n) = g(n) + d(n)*c(n) \qquad d(n) \text{ angle de déviation}$$

Angle entre deux vecteurs 3D 1- Vecteurs représentés par des coordonnées :

$$a = [x_a, y_a, z_a],\ b = [x_b, y_b, z_b]$$

$$\text{angle} = \arccos\left[(x_a * x_b + y_a * y_b + z_a * z_b) \not/ \left(\sqrt{(xa2 + ya2 + za2)} * \sqrt{(xb2 + yb2 + zb2)}\right)\right]$$

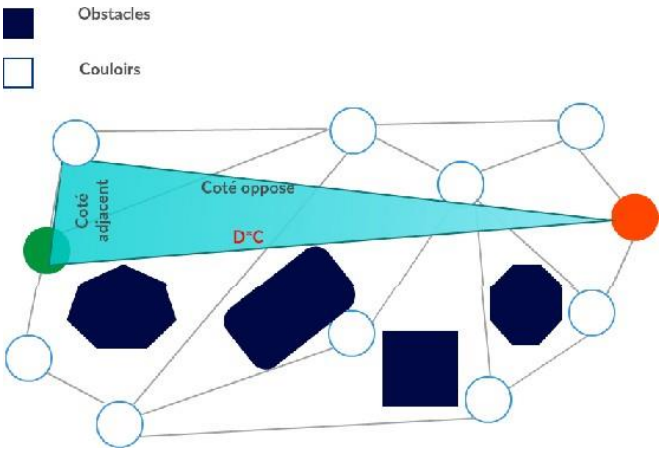


FIGURE 3.12 – Théorème de pythagore

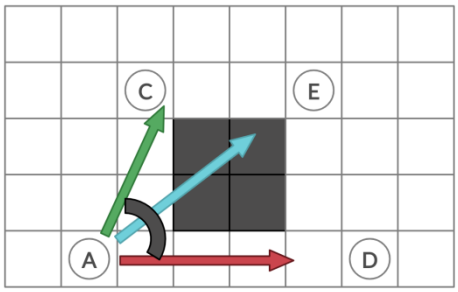


FIGURE 3.13 – Angle déviation minimale entre point départ et la destination

$$d(n) = \cos\Theta = \frac{U.V}{||U||||v||}, \quad \Theta = \cos^{-}1[\frac{U.V}{||U||||v||}]$$

Dans ce cas l'agent calcul l'angle de déviation entre le vecteur menant directement à la destination et les vecteurs conduisant à tous les nœuds adjacents au nœud dans lequel se trouve l'agent. Puis il choisit le plus petit angle. La même opération entre le nœud suivant et la destination jusqu'à atteindre le dernier nœud (destination).

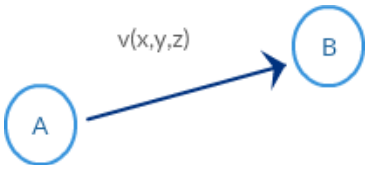


FIGURE 3.14 – Vecteur entre deux nœuds.

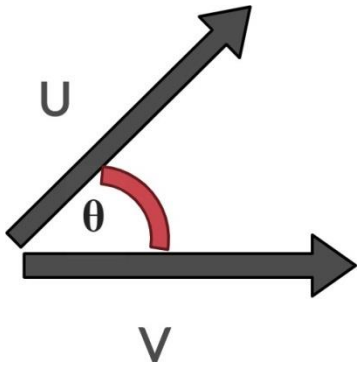


FIGURE 3.15 – Angle entre deux nœuds

La figure (3.14) représente le chemin suivi dans le cas (angle déviation minimale) sur environnement 3d.

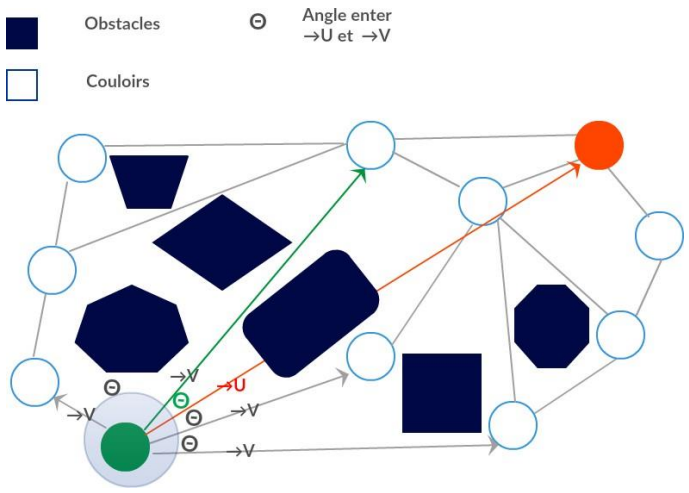


FIGURE 3.16 – Angle déviation minimale entre point de départ et la destination.

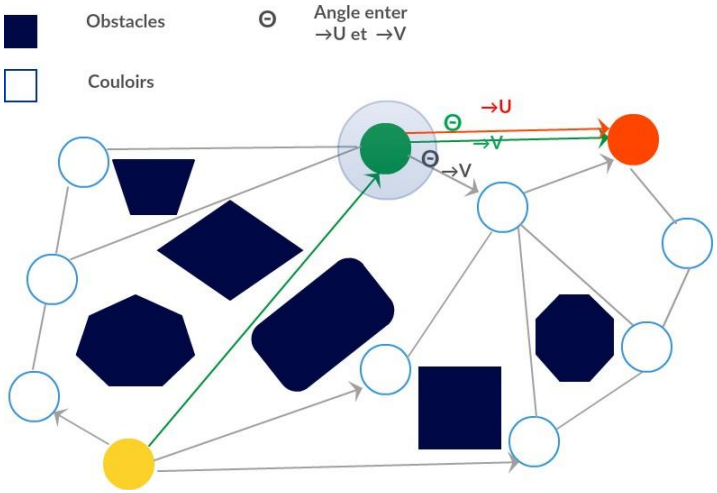


FIGURE 3.17 – Angle déviation minimale entre nœud actuel et le nœud de la destination.

Effort cognitif

Quand une personne recherche le chemin conduisant à sa destination, il fait un effort pour atteindre la destination et cela s'appelle effort cognitif, c'est-à-dire les connaissances de l'environnement dans lequel il se trouve ou non, autrement dit, plus le nombre de déviations est élevé, plus l'effort effectué augmente pour atteindre la destination. Dans ce critère, on veut réduire l'effort dépensé lors de la recherche du chemin, en réduisant le nombre de changements de direction. Dans ce cas, le chemin le plus court est le nombre de changement de direction à la destination.

Ce critère réduit le nombre de changement de direction de trajectoire, lorsqu' il y a moins de changement de direction, le chemin devient plus facile a trouvé.

On considère que l'agent ne connaît pas la région, il préfère le chemin qui contient moins de

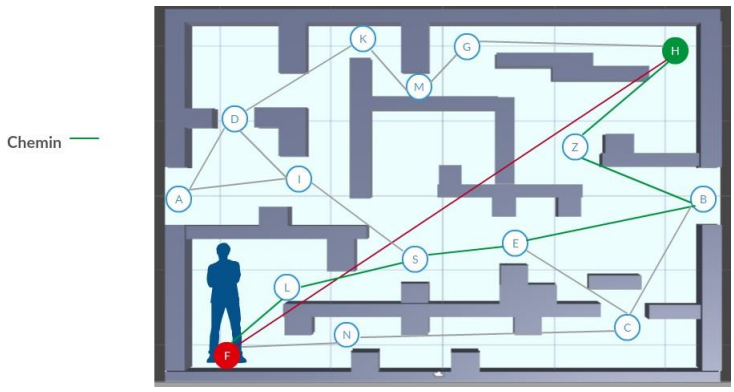


FIGURE 3.18 – Le chemin suivi dans le cas (angle déviation minimale)

changements de direction parce que cela lui économise les efforts lors de la recherche du chemin. Où l'algorithme parcourt tous les nœuds de graphes et il est nous retourne et nous montre le chemin qui contient le nombre le moins de déviations à partir du point de départ jusqu'à le point d'arrivée, on considère i chaque déviation est considérée à droite ou à gauche, son angle va devenir supérieur à 45 degrés ce qui dit il qu'il ya un changement de direction, l'agent suivra ce chemin.

-La formule de calcule le chemin deviendra :

$$f(n) = g(n) + d(n)*c(n) \qquad d(n) \text{ nombre de déviations()}$$

Si(Angle déviations >=45) d(n) = d(n) +1 // incrémentation le nombre de changement de direction

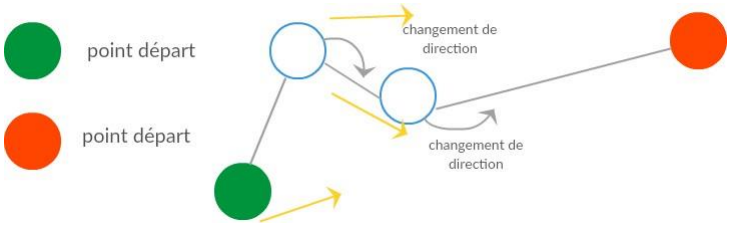


FIGURE 3.19 – Changement de direction.

La figure (3.19) montre comment l'algorithme trouve le chemin qui contient nombre minimale de changement de direction.

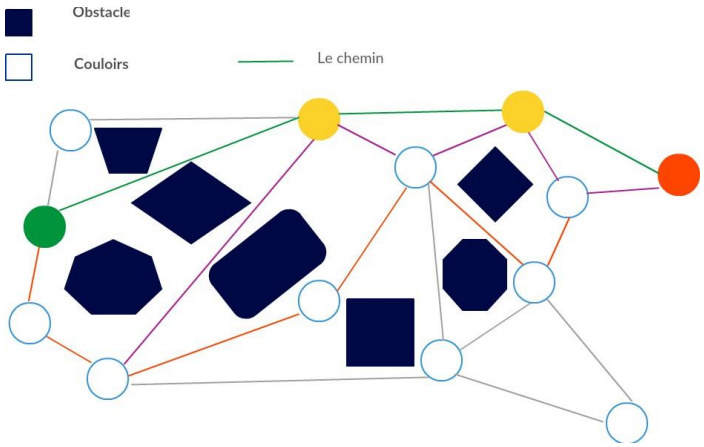


FIGURE 3.20 – Minimum de nombre de changement de direction.

La figure (3.20) illustre le chemin suivi dans le cas (nombre de changement de direction), on remarque qu'il a choisit le chemin qui contient le moins de nombre de déviation.

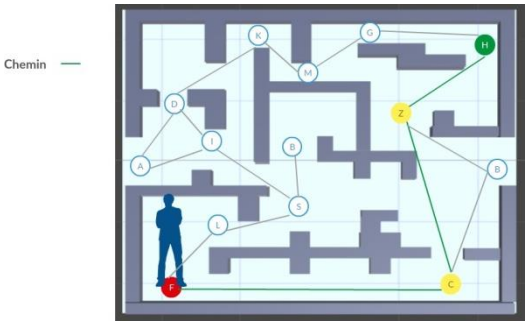


FIGURE 3.21 – Le chemin suivi dans le cas (nombre de changement de direction).

3.5 Conclusion

Ce chapitre nous a permis d'expliquer notre objectif et de proposer une conception globale et détaillée de notre système. Pour ce faire, nous nous sommes basés sur deux techniques intéressantes. La première consiste à la représentation de l'environnement par le maillage de navigation qui nous permet construire la surface navigable. La deuxième, elle consiste en l'exploitation de l'algorithme A-Etoile pour calculer le plus court chemin.

CHAPITRE 4

IMPLÉMENTATION ET RÉALISATION

Introduction

Après avoir fait l'étude conceptuelle de notre système établi, nous passons maintenant à l'implémentation de l'application, en présentant l'environnement et les outils utilisés pour arriver à notre objectif, comme l'environnement de développement et le langage de programmation ; nous avons utilisé ainsi que les procédures exploitées, par la suite nous allons présenter les résultats que nous avons obtenus.

La machine utilisée

Processeur	Intel (R) Core(TM) i5-8250U CPU @ 1.60GHz 1.8GHz
RAM	8.00 GB
carte graphique	- Intel(R) UHD Graphics 620 - NVIDIA GeForce MX130

C# est un langage de programmation orientée objet, commercialisé par Microsoft depuis 2002 et destiné à développer sur la plateforme Microsoft .NET. C'est un langage de programmation orienté objet dérivé de C et C ++ qui est utilisé pour développer des applications Web, ainsi que des applications de bureau, des services Web, des commandes,

unity est un moteur de jeu multiplateforme (smartphone, ordinateur, consoles de jeux vidéo et Web) développé par Unity Technologies. Il est l'un des plus répandus dans l'industrie du jeu vidéo, aussi bien pour les grands studios que pour les indépendants du fait de sa rapidité aux prototypages et qu'il permet de sortir les jeux sur tous les supports. Le logiciel a la particularité d'utiliser le langage de programmation (C charp) sur la plateforme.

Architecture globale de l'application

Notre application se compose de deux parties principales, la première est l'environnement dans lequel l'agent va interagir, et la seconde est l'agent qui naviguera dans l'environnement. Le système de navigation nous permet de créer des personnages qui peuvent naviguer dans le monde. Cela donne à nos personnages la possibilité de marcher dans leur environnement.

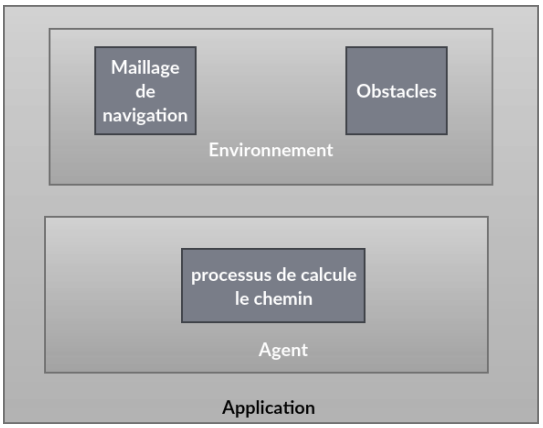


FIGURE 4.1 – Architecture global du l'application

Dans notre application, nous avons utilisé le système de navigation fourni par unity. Tout d'abord, notre application se compose de trois composants de base dans la construction de l'application, comme le montre la figure (4.1).

Le système de navigation nous permet de créer des personnages qui peuvent naviguer dans l'environnement. Cela donne à nos personnages la capacité de comprendre qu'ils doivent prendre des escaliers pour atteindre le deuxième étage ou sauter pour franchir un fossé. L'unité NavMesh se compose des éléments suivants :

— **Maillage de navigation (NavMesh)** (abréviation de Navigation Mesh maillage de navigation) est une structure de données qui décrit les surfaces praticables de l'environnement et permet de trouver le chemin d'un endroit praticable à un autre dans l'environnement. La structure de données est créée ou créée automatiquement à partir de la géométrie de notre niveau, nous l'expliquerons dans la section la construction de l'environnement.

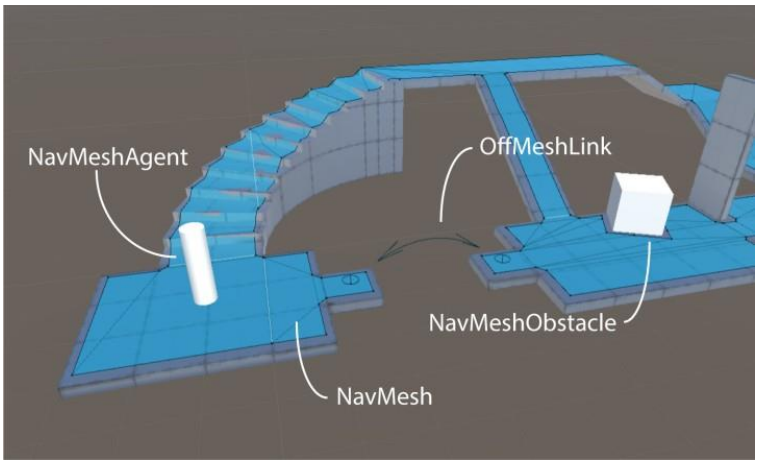


FIGURE 4.2 – Le système de navigation.

— **Le composant NavMesh Agent** nous aide à créer des personnages qui s'évitent tout en progressant vers leur objectif. Les agents raisonnent sur le monde de simulation en utilisant le NavMesh et ils savent comment s'éviter les uns les autres et déplacer les obstacles
— **Le composant NavMesh Obstacle** nous permet de décrire les obstacles mobiles que les agents doivent éviter lorsqu'ils naviguent dans le monde. Un tonneau ou une caisse contrôlée par le système physique est un bon exemple d'obstacle. Pendant que l'obstacle se déplace, les agents font de leur mieux pour l'éviter, mais une fois que l'obstacle devient stationnaire, il creusera un trou dans le navmesh afin que les agents puissent changer de chemin pour le contourner, ou si l'obstacle stationnaire bloque le chemin façon, les agents peuvent trouver un itinéraire différent.

Environnement

Dans cette partie nous préparons l'environnement 3d dans lequel l'agent se déplacera pour trouver son chemin. Notre environnement 3d' est une petite ville qui contient des surfaces navigables et des obstacles statiques et dynamiques. La figure 4.3 représente notre environnement de simulation.

Surface navigable

Pour définir la zone navigable, on relie notre environnement au composant (navmesh) maillage de navigation.
Le NavMesh stocke cette surface sous forme de polygones convexes. Les polygones convexes sont une représentation utile, car nous savons qu'il n'y a aucun obstacle entre deux points à

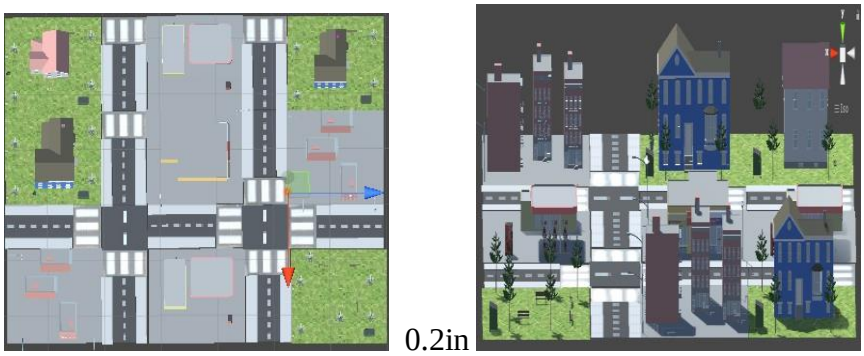


FIGURE 4.3 – Environnement de simulation

l'intérieur d'un polygone. En plus des limites des polygones, nous stockons des informations sur les polygones voisins les uns des autres. Cela nous permet de raisonner sur toute la zone de marche. Le maillage de navigation (NavMesh) est une structure de données spécialement conçue pour prendre en charge la planification d'itinéraire et les calculs de navigation. Il code une décomposition convexe de la scène dans laquelle chaque polygone convexe (nœuds) est une zone piétonnable et connectée à l'aide de liaisons (arêtes) qui fournissent un contact inter cellulaire aux agents.
La fonction principale du réseau de navigation est de représenter efficacement l'environnement libre. Pour permettre le calcul des requêtes d'itinéraire à des moments optimaux et pour prendre encharge d'autres requêtes spatiales utiles pour la navigation.

La figure 4.4 représente notre environnement de simulation après l'application du maillage de navigation. On remarque que il y a des divisions dans la surface navigable, sur laquelle en compte comme un Coût de la traverser dont on considère comme un critère dans le processus de calcule le plus court chemin.

Obstacles

L'agent peut circuler dans l'environnement, d'un point de départ donné à un but donné. Il suit ainsi le chemin le plus court et évite les collisions avec des obstacles ou avec d'autres agents.

FIGURE 4.4 – Environnement de simulation après l'application de maillage de navigation.

Obstacles statiques

Pour créer des obstacles statiques, nous utilisons la composante NavMesh Obstacle, ce qui signifie que nous allons percer des trous dans le maillage de navigation. Comme il apparaît sur la figure 4.5.

FIGURE 4.5 – Obstacles statiques.

Comme le montre la figure, la zone bleue est la zone navigable et la zone blanche sont des obstacles.

L'agent considère ces trous comme une zone non navigable et les évite lors de ses déplacements.

Obstacles dynamiques

Les obstacles dynamiques sont des personnes qui se déplacent dans l'environnement tout en exécutant une application qu'un agent évite lors de ses déplacements.
Dans cette application nous avons représenté ces obstacles sous formes de caractères (personnes) se déplacent dans l'environnement vers des destinations aléatoires et qui sont programmés d'une manière (way point).

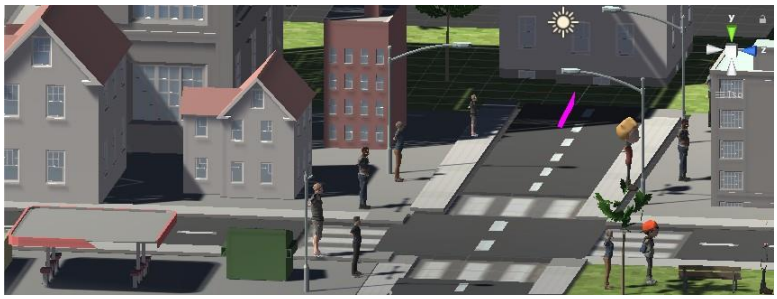


FIGURE 4.6 – Obstacles dynamiques.

```
public class ObstacleDynamic : MonoBehaviour {

public GameObject[] points;
Animator obsAnim;
NavMeshAgent obs;
void Start () {
obs = GetComponent<NavMeshAgent>();
obsAnim = GetComponent<Animator>();
int i = Random.Range(0,points.Length);
obs.SetDestination(points[i].transform.position);

}

// Update is called once per frame
void Update () {
if(obs.remainingDistance<= 0.5f)
{
int i = Random.Range(0,points.Length);
obsAnim.SetBool("isWalking",true);
obs.SetDestination(points[i].transform.position);
}
```

```
}
}
```

— points : c'est un tableau qui contient des points, ces points ont des coordonnées (x,y) dans l'environnement, où les obstacles dynamiques se dirigeront vers ces points.
— i : est une variable entiere aléatoire. Elle prend une valeur aléatoire entre zéro et la longueur de la matrice, c'est-à-dire entre zéro et le nombre de points dans le tableau.
— void Start() : est une fonction pour initialiser les variable et les appeler les composantes (GameObjects).
— void Upadate() : la mise à jour s'exécute cette fonction à chaque frame. Où nous appelons à l'intérieur les fonctions que nous voulons implémenter dans chaque frame.
— remainingDistance() :La distance entre la position de l'agent et la destination sur le chemin actuel. Si la distance restante est inconnue, cela aura une valeur de l'infini, distance restante à chaque image, j'obtiens Infinity pour une partie du chemin jusqu'à ce qu'il commence à renvoyer des réels (21.21864, 21.0846, 20.95449 ...) jusqu'à ce qu'il se rapproche le plus possible de la destination, ce qui renvoie 0. nous avons spécifié distance restante par 0.5

Agent

Nous lions le personnage au composant (NavMesh Agent) pour contrôler l'agent via le script.

Cette composante va permettre à l'agent de se déplacer. Un agent ne peut se déplacer dans l'environnement si elle possède différents paramètres que nous définissons et manipulons à travers des scripts.

FIGURE 4.7 – Agent.

```
private void ClickToMove()
```

```
{
Ray ray =  Camera.main.ScreenPointToRay(Input.mousePosition);
RaycastHit hit;
bool hasHit = Physics.Raycast(ray, out hit);
if(hasHit)
{
SetDestination(hit.point);
}
if(Vector3.Distance(agent.destination, transform.position)<= agent.stoppingDistance
{
agentAnimator.SetBool("isWalk",false);
}
if(agent.hasPath)
{
TracePath();
}

}
```

Grâce à cette fonction nous déterminons la destination par la souris et déplacons l'agent vers la destination.

- ray, hit : détermine le point d'arrivée. L'idée est tout ce que nous cliquons sur la scène, nous projetons le rayon de la caméra vers la scène sur le point sur lequel on a cliqué. Le rayon entre en collision avec la scène à travers cette collision, nous obtenons le point avec des coordonnées(x,y,z) vers lequel se déplace l'agent.
- hasHit : Nous le testons si nous cliquons sur la scène ou non.
- SetDestination() :cette fonction prend le paramètre du vecteur(x,y,z) du point cible pour se déplacer. booléen Vrai si la destination a été demandée avec succès, déclenchant ainsi le calcul d'un nouveau chemin. Renvoie l'ensemble de destinations pour cet agent.
  - Si une destination est définie mais que le chemin n'est pas encore traité, la position renvoyée sera une position de navmesh valide la plus proche de la position précédemment définie.
  - Si l'agent n'a pas de chemin ou de chemin demandé - renvoie la position des agents sur le navmesh.
  - Si l'agent n'est pas mappé au navmesh (par exemple, la scène n'a pas de navmesh) - renvoie une position à l'infini.
  - S'il n'est pas possible de trouver une position de navmesh valide à proximité (par

exemple, la scène n'a pas de navmesh), aucun chemin n'est demandé. Nous utilisons SetDestination pour vérifier la valeur de retour si nous devons gérer ce cas explicitement.

```
void  TracePath()
{
if(agent.path.corners.Length < 2)
{
return;
}

for(int i=1; i< agent.path.corners.Length;i++)
{
Vector3 pointPosition =  new Vector3(agent.path.corners[i].x,agent.path.corners[i].
tracePath.SetPosition(i,pointPosition);
}
}
```

— TracePath() : cette fonction trace le chemin par lequel l'agent passe.

Dans cette section nous allons tester notre application et analyser les résultat de ces tests.

Processus de déroulement de la simulation

La figure 4.8 représente le processus de déroulement de l'application, d'abord le choix de l'environnement, ensuite déterminer le point départ et le point d'arrivée, enfin on obtient le résultat.

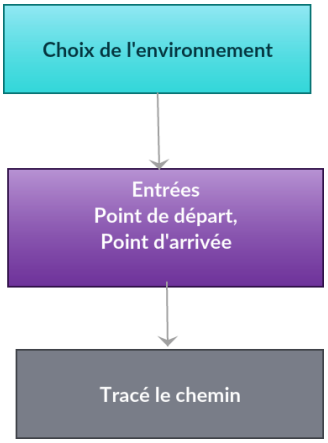


FIGURE 4.8 – Processus de déroulement la simulation

Choix de l'environnement

En choisissant l'environnement dans lequel l'agent évolue. Nous déterminons où se trouvent les obstacles.

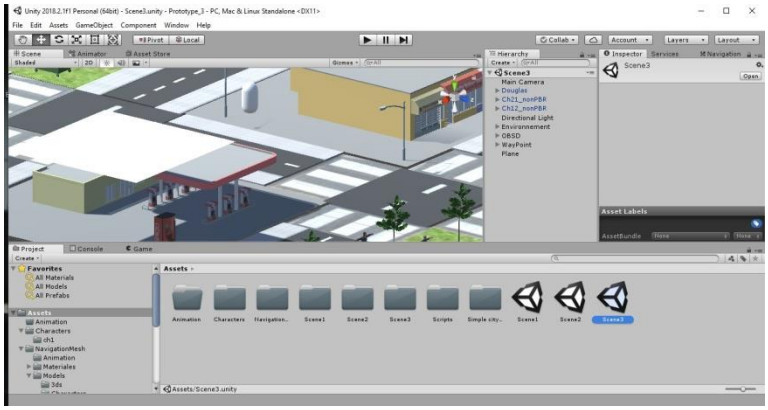


FIGURE 4.9 – Choix de l'environnement.

Ensuite, nous attribuons à chaque zone le coût de la traversée.

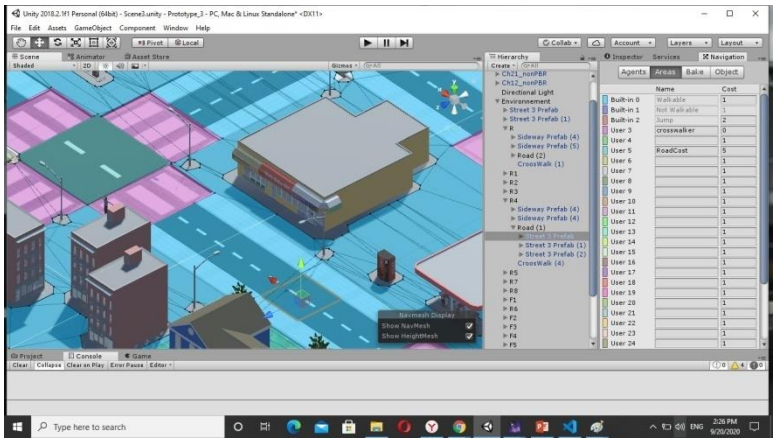


FIGURE 4.10 – Le coût du zones.

Les entrées

— Le point d'arrivée, nous définissons le point d'arrivée en cliquant avec notre souris sur la surface d'environnement où nous voulons que l'agent aille.
— Le point de départ que nous définissons en plaçant l'agent à l'endroit à partir duquel nous voulons qu'il commence à calculer son chemin.

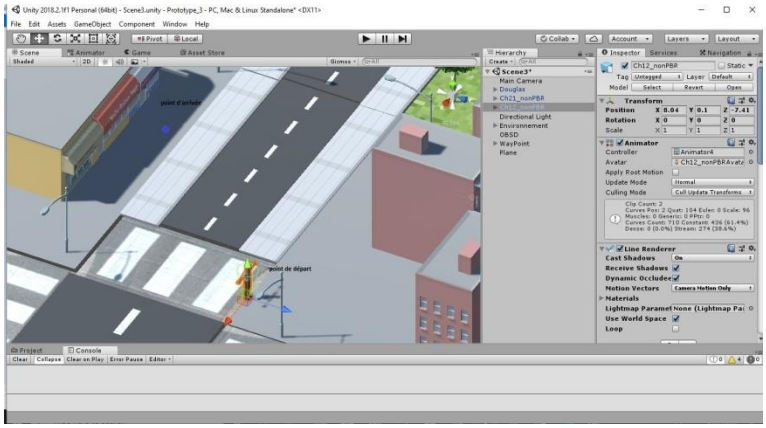


FIGURE 4.11 – Les entrées point de départ, point d'arrivée.

Tests

Test 1

Dans ce test nous expérimentons la distance euclidienne et on considère le coût de la traversée de toutes zones est égale, dans le processus de recherche du chemin le plus court.

Entrées POINT DE DÉPART
POINT D'ARRIVÉE
CHOIX D'ENVIRONNEMENT
DISTANCE EUCLIDIENNE

Résultat Dans le premier test et en se basant uniquement sur la distance euclidienne pour

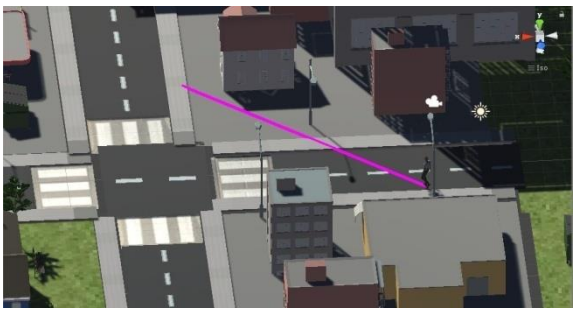


FIGURE 4.12 – Test1 distance euclidienne

calculer le chemin, on remarque que l'agent suivra le chemin le plus direct vers la destination.

Test 2

Nous faisons la même première expérience en utilisant la distance euclidienne dans le processus de recherche du chemin le plus court avec le changement du coût de la traversée des zones. On remarque que l'agent évitera le transit sur les zones de coûts plus élevés.

Entrées

POINT DE DÉPART
POINT D'ARRIVÉE
CHOIX D'ENVIRONNEMENT
DISTANCE EUCLIDIENNE
COÛT DE LA TRAVERSÉE DU ZONES
- **MARCHES À PIETON = 0**
- **ROUTE = 5**
- **TROTTOIR = 1**

OBSTACLES
- **OBSTACLES STATIQUES**

FIGURE 4.13 – Test 2.

Dans le deuxième test après avoir ajouté le coût de la traversée pour chaque région, on remarque que l'agent évite les zones les plus chères et il choisira le chemin le plus court.

Test 3

Nous faisons la même deuxième expérience en utilisant la distance euclidienne dans le processus de recherche du chemin le plus court et en changeant le coût de traversée des régions et en ajoutant des obstacles dynamiques. Ici, l'agent évitera les obstacles dynamiques en se dirigeant vers sa destination.

Entrées

POINT DE DÉPART
POINT D'ARRIVÉE
CHOIX D'ENVIRONNEMENT
DISTANCE EUCLIDIENNE
COÛT DE LA TRAVERSÉE DU ZONES
- **MARCHES À PIETON = 0**

- **ROUTE = 5**
- **TROTTOIR = 1**

OBSTACLES

- **OBSTACLES STATIQUES**
- **OBSTACLES DYNAMIQUES**

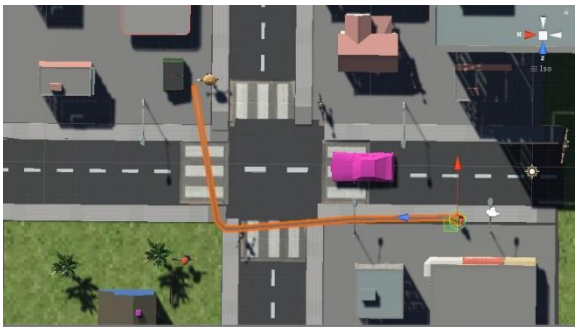


FIGURE 4.14 – Test 3

Dans le troisième test après avoir ajouté des obstacles dynamiques, il ralentit sa vitesse parfois, on remarque que lorsque l'agent se rend à sa destination, son chemin est affecté par des obstacles dynamiques et il changera son chemin parfois.

Discussion des résultats

Dans ce chapitre, nous montrons les résultats obtenus dans différents types de distance (distance euclidienne, angle déviation, nombre changement de direction). On remarque que le choix de l'angle de déviation lors de la recherche du chemin le plus court est plus réaliste en suivant le chemin, c'est-à-dire qu'il est plus proche de la pensée humaine, l'homme ne peut pas deviner avec précision la distance dans le cas de la distance euclidienne, mais suit plutôt le moindre angle de déviation entre lui-même et sa destination. En revanche, on remarque que plus le chemin a le moins de changement de direction, plus il sera rapide et facile de trouver une destination.

Conclusion

Au terme de ce dernier chapitre, nous avons repris la démarche entreprise pour la réalisation de notre projet.

Tout d'abord, nous avons présenté notre environnement de travail et les outils de programmation utilisés avant de passer à la phase d'implémentation, en commençant par la modélisation

de l'environnement par les maillages de navigation.

Ensuite, nous sommes passés à la phase recherche de chemin .Dans lequel nous nous sommes appuyés sur l'algorithme A-étoile pour calculer le chemin le plus court, qui est implicitement appliqué dans le système de navigation.

Finalement, nous avons terminé par la présentation de notre réalisation de notre travail.

Conclusion générale

Ce mémoire de fin d'études a eu pour objectif de réaliser de la simulation du comportement humain lors de la recherche de son chemin, et les critères que doit prendre en considération l'agent virtuel dans le choix et la recherche de son chemin tel qu'un être humain.

Nous avons commencé dans le premier chapitre par une introduction et une étude sur le domaine de l'animation comportementale, modélisation du comportement des humanoïdes virtuels et tout ce qui concerne leurs caractéristiques. Nous avons présenté la boucle de décision et les modèles de comportement, représentation de l'environnement dans lequel navigue notre agent virtuel.

Dans le deuxième chapitre nous avons fait une étude de l'état de l'art. On a présenté les algorithmes de recherche du plus court chemin qui utilisent les graphes dans leur processus de calcul, où elles dépend sur la distance, puis on a passé à l'optimisation multi-objectifs où nous avons donnée de la définition de la fonction de coût et présenter les méthodes de résolution.

Nous avons proposé dans le troisième chapitre la conception de notre système. Nous avons divisé notre system en deux parties. Dans la première partie l'environnement qui contient la représentation de la surface navigable et les obstacles, et la deuxième la technique avec laquelle l'agent calcule son chemin.

Le dernier chapitre est destiné à l'implémentation de notre travail, où nous avons montré les différentes étapes suivies pour développer notre projet.

En conclusion dans le quatrième chapitre, on a réalisé l'application et présenter les résultats des tests de notre application, puis on a fait le diagnostic de ces résultats.

Enfin nous avons constaté qu'il ya plusieurs critères qui sont pris en considération dans le choix du plus court chemin.

BIBLIOGRAPHIE

[1] C. Mars, “Peuplement automatisé de bases géométriques urbaines,” 2006.

[2] C. Foudil, *Animation comportementale : simulation de foule d’humains virtuels*, 01 2010.

[3] F. Lamarche and S. Donikian, “Crowd of virtual humans : a new approach for real time navigation in complex and structured environments,” in *Computer graphics forum*, vol. 23, no. 3. Wiley Online Library, 2004, pp. 509–518.

[4] H. Whiston, “Multi-objective pathfinding in dynamic environments,” 2018.

[5] L. SAADI, “Optimisation multiobjectifs par programmation génétique,” Ph.D. dissertation, Université de Batna 2, 2007.

[6] N. Gunantara, “A review of multi-objective optimization : Methods and its applications,” *Cogent Engineering*, vol. 5, no. 1, p. 1502242, 2018.

[7] A. Berro, “Optimisation multiobjectifs et stratégies d’évolution en environnement dynamique,” Ph.D. dissertation, ANRT [diff.], 2001.

[8] V. Barichard, “Approches hybrides pour les problèmes multiobjectifs,” Ph.D. dissertation, Université d’Angers, 2003.