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 : IVA14/M2/2021

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **Image et Vie Artificielle (IVA)**

Simulation du Mouvement des Piétons Dans un Bâtiment

Par :

AMRAT NISRINE

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

Nom Prénom	grade	Président
BOUCETTA MEBAREK	MAA	Rapporteur
Nom Prénom	grade	Examinateur

Année universitaire 2020-2021

Remerciements

Tout d'abord, je remercie Dieu d'avoir accompli ce travail, j'exprime ma profonde gratitude et mes sincères remerciements à mon encadreur monsieur BOUCETTA MEBARAK pour le temps qu'il m'a consacré et son suivi pendant la période de réalisation de mon projet.

Je remercie également les membres du jury pour leur honneuret leur appréciation pour leur participation et leur discussion, ma thèse.

Dédicaces

Je dédie ce travail

À mon marie et mes chers parents pour leurs conseils, encouragements et soutien sans fin en signe de ma gratitude et de mon affection,en espérant qu'ils sont fiers de moi.

À mes chers frères et sœurs.

À tous mes amis et mes collègues de travail.

Table des matières

Chapittre 02 La simulation comportementale

Chapittre 03 Conception

Chapittre 04 Implémentation

Table des Figures

Table des Tableau

Résumé

Dans la vie quotidienne les mouvements sont le moyen principale d'interagie avec l'environnement, ces interaction parfois provoque des collisions et des chevauchent surtout dans les cas de panique, l'animation comportementale est la méthode la plus utilisée pour simuler le mouvement des humains dans un monde virtuel. Cette méthode donne à l'humain virtuel l'aspect d'autonomie et lui permettre de perçoive son environnement, agisse sur ce dernier et surtout prenne mêmes des décisions en rapport avec la situation perçue dans le but d'exhiber un comportement cohérent proche de l'organisme vivant simulé.

Dans le cadre de ce mémoire, nous souhaitons apporter de contributions à la simulation du mouvement des piétons dans un bâtiment de manière réaliste. Cette situation nécessite une analyse rigoureuse du comportement des individus via la simulation ainsi que la description et la discrétisation de l'environnement pour permettre à l'homme virtuel (appeler agent virtuel) de ce déplacé et évité les obstacles et les collisions dans le même niveau hiérarchique de bâtiment ou bien entre les déférents niveaux hiérarchique.

Nous nous sommes intéressés à développer les aspects suivants : la discrétisation de la scène, la planification de chemin (la recherche de chemin) dans un environnement virtuel constitué de plusieurs étages; l'évitement des obstacles et de collisions entre les piétons pendant le déplacement; et enfin l'application des règles sociales de mouvement de l'individu.

Mots-clés :

Animation ,Planification de chemin, comportement, évitement de collision, navigation, simulation ,Environnement, agent virtuel,

ملخص

في الحركات الحياتية اليومية هي الوسيلة الرئيسية للتفاعل مع البيئة ، وهذه التفاعلات تسبب أحيانا اصطدامات وتتداخل خاصة في حالات الذعر ، والرسوم المتحركة السلوكية هي الطريقة الأكثر استخداما لمحاكاة حركة البشر في العالم الافتراضي .وهذه الطريقة تعطي الإنسان الافتراضي جانب الاستقلالية وتتيح له أن يدرك بيئته ، ويتصرف وفق هذا الأخير ، وقبل كل شيء يتخذ قرارات فيما يتعلق بالوضع المتصور بهدف إظهار سلوك متماسك قريب من الكائن الحي المحاكي.

وفي إطار هذا الموجز ، نود أن نسهم في محاكاة حركة المشاة في مبنى بطريقة واقعية ويتطلب هذا الوضع تحليلاً دقيقاً لسلوك الأفراد من خلال المحاكاة فضلاً عن وصف البيئة وتشويهها للسماح للرجل الافتراضي (المسمى العامل الافتراضي) بتجنب العقبات والاصطدامات في نفس مستوى البناء الهرمي أو بين لمستويات الهرمية الانحرافية.

ونحن مهتمون بتطوير الجوانب التالية: تشويه المشهد ، وتخطيط المسار (البحث عن المسار) في بيئة افتراضية تتألف من عدة طوابق ؛وتجنب العقبات والاصطدامات بين المشاة أثناء السفر ؛ وتطبيق القواعد الاجتماعية لحركة الفرد.

كلمات البحث:
الرسوم المتحركة ، تخطيط المسار ، السلوك ، تجنب الاصطدام ، الملاحة ، المحاكاة ، البيئة ، الوكيل الافتراضي.

Summary

In everyday life movements are the main means of interacting with the environment, these interactions sometimes cause collisions and overlap especially in cases of panic, behavioral animation is the most widely used method to simulate the movement of humans in a virtual world. This method gives the virtual human the aspect of autonomy and allows him to perceive his environment, act on the latter and above all make decisions in relation to the perceived situation with the aim of exhibiting a coherent behavior close to the simulated living organism.

Within the framework of this brief, we wish to make contributions to the simulation of pedestrian movement in a building in a realistic manner. This situation requires a rigorous analysis of the behavior of individuals via simulation as well as the description and the discretization of the environment to allow virtual man. to avoid obstacles and collisions in the same hierarchical building level or between the deferential hierarchical levels.

We are interested in developing the following aspects: the discretization of the scene, the planning of the path (the search of path) in a virtual environment consisting of several floors; avoiding obstacles and collisions between pedestrians while on the move; and enforcing the social rules of movement of the individual

Keywords:
Animation ,Path planning, behavior, collision avoidance, navigation, simulation ,Environment, virtual agent,

Introduction général

Introduction général

Les améliorations techniques rapides dans le monde de l'informatique conduit à l'émergence de nombreuses sciences moderne, est l'une des disciplines utilisée dans des domaines divers qui vont du divertissement à des usages professionnels de haute précision; l'animation comportementale qui est la visualisation de la vie réel d'une façon virtuel consiste à la simulation des comportements provoquer par l'interaction entre des personnages virtuels dans son environnements virtuels lui aussi.

La simulation des comportements de l'humain virtuel possède des contraintes associées à la structure de l'environnement, où des informations provenant de ce dernier son percevoir par l'individu qui va prendre des décisions en rapport avec la situation perçue et fait des réactions, c'est une simulation cohérente proche de l'organisme vivant, le niveau d'intelligence et d'autonomie dépend de la nature de l'humain virtuel, et les forces sociologies qui affecté sur ces comportements. Toute cette contrainte dépend aux choix de déférentes méthodes et modèles utilisé dans la simulation.

Nous nous intéressons dans ce travail de la simulation de mouvement des agents virtuels dans un bâtiment, ou il y a un environnement constituée de plusieurs niveaux verticaux et peupler de quelques agents qui se déplace à l'intérieure de chaque niveau ainsi qu'entre les niveaux verticalement dans ce bâtiment. Ce qui va produire des interactions pendant le déplacement où le niveau d'autonomie de l'agent virtuel permettre de représentée des actions plus réalistes nécessite l'amélioration de la crédibilité et de l'intelligence de l'agent.

Nous sommes intéressés de développer les aspects suivants : déplacement des piétons autonome dans un environnement Virtual composé de plusieurs niveau vertical, la planification de chemin (la recherche de chemin) dans un environnement virtuel ; l'évitement de collisions avec des obstacles statiques ou dynamiques, toutes en réalisons une simulation réaliste.

Ce mémoire est structuré en quatre chapitres de la manière suivant :

Le premier chapitre « l'animation comportementale », nous présentons la boucle d'animation comportementale, les différents types des représentations d'environnement virtuel, les types d'agents virtuels, et enfin les comportements de l'agent virtuel.

Dans le deuxième chapitre, « la simulation comportementale », présente les méthodes de simulation comportementale et les algorithmes de parcoure de chemin.

Le troisième chapitre présent le cycle de développement de notre modèle proposé.

Le dernier chapitre présent notre environnement de développement avec les outils utilisés pour l'implémentation de notre application, nous avons également présenté quelques images de scénarios pour notre simulation.

Et nous avons terminé par une conclusion générale qui résume notre système.

.

Chapitre 01:

Animation comportemental

1. Introduction

Le développement des techniques de modélisation et d'animations nous a permis de crées des mondes virtuel similaire au monde réel sa fait une révolution dans le domaine des effets visuels, ces techniques sont employée par l'industrie du cinéma, du jeu vidéo ainsi que des nombreuses applications de réalité virtuelle .

La réalisation d'une animation nécessite une analyse rigoureuse du comportement des individus dans son environnement et l'utilisation des méthodes pour atteindre des résultats similaires et plus réaliste.

Dans ce premier chapitre, on va présenter la définition de : l'animation comportementale, environnement virtuel et agent virtuel puis on va présenter aussi les domaines d'application ainsi que les différentes méthodes utilisées pour la représentation de ces composants.

2. Définition de l'animation comportementale

L'animation comportementale fait allusion aux différentes méthodes dont l'informatique se sert afin de modéliser les comportements des entités qui peuplent un monde virtuel et qui ont comme caractéristiques communes la capacité de percevoir ce monde ainsi qu'un certain niveau d'autonomie. En général, toutes les approches proposées par l'animation comportementale ont comme finalité de sélectionner la meilleure action à réaliser dans une certaine situation parmi l'ensemble de toutes les actions possibles. Si l'on est capable de bien réaliser cette sélection, on peut modéliser toute sorte d'individus vivants (plantes, animaux et êtres humains) [1] .

Donc L'animation comportementale modélise le comportement des personnages appelés aussi agents, acteurs ou autonomes (des êtres vivants) qui évoluent dans un environnement lui-même virtuel et qui répond à des stimuli de l'environnement, elle tente de modéliser le processus de raisonnement d'un personnage synthétique autodirigé par spécifier la décision du système: en fonction des stimuli qu'il perçoit, il agit.

3. Les domaines d'applications de l'animation comportementale

L'application de l'animation comportementale est extrêmement large presque dans toutes les domaines de la vie, et voici une description brèves de quelques domaines.

3.1. Jeux vidéo

Ce domaine est parmi les premiers et les plus classés dans l'utilisation des environnements virtuels 3D et selon le type de jeux seront le type de d'environnement et sa nature, pour cela on va citer quelques types de jeux. [2]

- **les jeux de tirs subjectifs ou FPS (First-Person Shooters) :** le point de vue est celui de l'utilisateur qui est, en général, un personnage muni d'une arme. (Exemple : Quake, Unreal Tournament, Half-Life, ...). Dans ces jeux, le monde est visuellement très réaliste et relativement étendu.
- **les jeux de plateforme :** ce sont les jeux dont l'action est basée sur un personnage tierce (Exemple : Space panic, Donkey Kong, Super-Mario Brothers, ...). Pour ce type de jeu, le monde n'est pas toujours tridimensionnel, son réalisme importe peu et il est plutôt restreint.
- **les jeux de combat :** ces jeux sont généralement jouables à deux et les deux personnages doivent s'affronter au cours d'un combat (Exemple : Soul Calibur, Tekken, Mortal kombat, ...). Le monde y est restreint et le réalisme est de moindre importance.
- **les jeux de course :** ces jeux se déroulent sur un circuit et le joueur doit concourir à une courses de voiture ou tout autre type de véhicule (Exemple : Grand Turismo, Need For Speed,...). Pour ce type de jeux, la superficie du monde se limite aux circuits mais le réalisme visuel et physique y est généralement essentiel (Figure 1.1)

Figure 1.1. Les différents types de jeu : un jeu de combat (1.1c), un jeu de course (1.1d), un jeu de rôle (1.1h) et un jeu de sport (1.1i).

3.2. Les Métavers

Les métavers sont des mondes virtuels 3D dans lesquels les utilisateurs évoluent sous forme d'avatars personnalisables. Dans ces environnements, l'immersion se fait entièrement via l'avatar de l'utilisateur. De plus, le contenu des métavers est créé par les utilisateurs eux-mêmes. Ceci implique un nombre considérable de contenus sans cesse en évolution au gré des créations et des modifications des utilisateurs. La fonction principale d'un métavers est de mettre en relation des utilisateurs réels du monde entier par l'intermédiaire de leurs avatars afin qu'ils y développent des activités éventuellement lucratives. Les utilisateurs forment une communauté au sein de laquelle ils nouent les mêmes relations que dans le monde réel. Les métavers les plus importants sont devenus des micro-sociétés, ainsi, les plus développées ont mis en place un commerce virtuel ou v-commerce associé à une monnaie virtuelle. [2]

3.3. L'apprentissage En Ligne (E-Learning) Et Les Jeux Sérieux

Les environnements virtuels ne sont pas utilisés uniquement dans le domaine du divertissement, ils servent de plus en plus à des fins d'apprentissage par la simulation. C'est ce qui est désigné depuis peu sous le terme d'"e-learning". Le domaine de l'éducation n'est pas nouveau. Dès 1973, les jeux éducatifs The Oregon Trail 8 et Lemonade Stand 9 ont été développés dans ce but.

La définition d'un jeu sérieux varie selon les auteurs, certains considèrent que celui-ci implique obligatoirement la mise en place d'une règle du jeu ainsi que d'indications sur la façon d'y jouer à destination de l'utilisateur (ce que l'on résume souvent par l'utilisation de l'anglicisme "gameplay"). D'autres emploient ce terme pour désigner tout type de simulation à but d'apprentissage. Les jeux sérieux intéressent de plus en plus de domaine depuis le sucée de jeux de simulation d'entrainement militaire et de mission de combats « America 's Army » afin d'inciter les joueurs à s'engager dans l'armé américain, il a été lancé en 2002 et totalisé 17 Millions de téléchargement en 2004. [2]

Figure 1.2. America's Army

3.4. Le Web Géospatial

Les environnements virtuels 3D sont également représentés dans ce que l'on peut appeler le web géospatial. Cette dénomination regroupe tous les services de cartographie 3D disponibles sur le web permettant de visualiser des données géolocalisées issues des systèmes d'information géographique ou SIG. On peut classer dans cette catégorie Google Earth.(Figure 1.3) [2]

(b) Google Earth

Figure 1.3. Vues de New York dans Google Earth

4. Les composantes de l'animation comportementale

L'animation comportementale est basée sur interaction des agents intelligents autonomes avec son environnement virtuel et les comportements d'un individu qui signifient l'ensemble des actions et réactions observables de ce dernier dans son environnement pour obtenir une animation plus réaliste. Donc on peut distinguer trois composantes à modélisé pour atteindre ce but :

4.1. Environnement virtuel

L'environnement est dans le quelle les entités évoluent il peut être considéré comme un modèle trois dimensions d'environnements réels ou imaginaires que l'on peut visualiser et avec lesquelles on peut interagir en temps-réel [1] Des informations sensorielles complémentaires (sonores, tactiles, ...) peuvent venir enrichir ce modèle.

4.2. Agent virtuel

Un agent est une entité active, autonome mais sociable, qui peut être intelligente et qui interagit avec un environnement dynamique (son comportement dépend du contexte).Un agent minimal fonctionne selon un cycle de perception, de décision et d'actions et dispose d'une connaissance de son état interne. [3]

4.3. La simulation comportementale

Consiste la réalisation du scénario et le trame de l'action. La simulation comportementale est le moyen de faire interagir de manière naturelle des acteurs en simulant leurs capacités dans un environnement.

5. La boucle de l'animation comportementale

L'animation comportementale est basé sur trois phases forment une boucle d'animation comportementale (Figure 1.4) L'interaction d'un personnage avec son environnement est capable de montrer comme résultat un comportement adaptatif, réaliste et efficace.

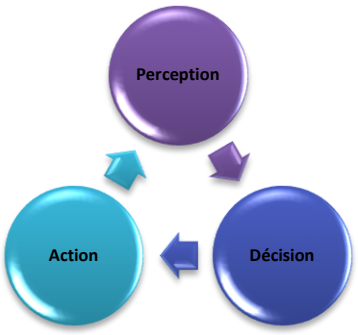


Figure 1.4. La boucle d'animation

Dans ce système les fonctions de chaque composant sont :

- Perception : à travers les sens, tant l'humain que l'agent perçoit l'environnement et, si c'est le cas, traduisent ces perceptions en signaux pour le module de décision.
- Décision : en utilisant l'information provenant du module de perception, le système décide quoi faire et quand le faire. Cette décision est prise en prenant en compte les motivations, les buts, les désirs, etc.
- Action : ce module décrit comment exécuter l'action ou plan produit par le module de décision. Ici, les plans sont traduits en actions concrètes comme se déplacer, se nourrir, tourner, etc. [4]

6. Représentation de l'Environnement virtuel

Environnement virtuel ou monde virtuel est l'espace d'interaction dans un monde artificiel imaginaire à trois dimensions créé par ordinateur. D'une manière générale, un environnement est un espace, muni d'une topologie ou non, possédant des objets passifs ou actifs, et des lois d'interactions entre lui-même et ses objets.

6.1. Les niveaux de l'information

Les différents types de modèles de l'environnement virtuel sont possibles selon les informations que l'on souhaite exploiter. On peut distinguer les niveaux d'information présents dans la littérature en trois grands types [5]:

- le niveau géométrique
- le niveau topologique
- le niveau sémantique.

6.1.1 Modélisation géométrique

Dans la représentation de l'espace libre, Il existe deux grandes familles de techniques existent : la décomposition de l'espace libre en cellules géométriques et les cartes de cheminement (Figure 1.5.)

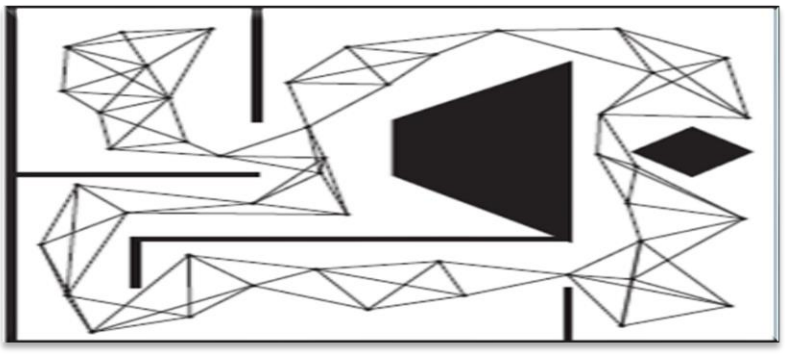


Figure 1.5. Carte de route probabiliste (PRM)

6.2.2 Modélisation topologique

Le graphe est l'outil commun pour les modèles topologiques, il est utilisés même si d'autres outils peuvent être mis en œuvre conjointement ou non (modèles de réseaux ou hiérarchiques) mais les nœuds (représentant les lieux) comme les arcs (représentant la connectivité entre les lieux) qui le constituent peuvent être de différentes natures. [5]

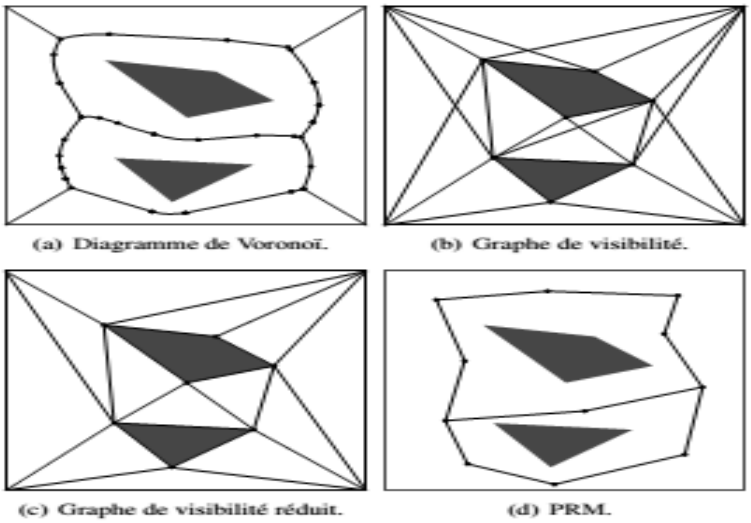


Figure 1.6. Modélisation topologique

6.1.3 Modélisation sémantique

Le modèle sémantique vient comme une sur-couche du modèle topologique. Une démarche incrémentale entre la modélisation topologique et la modélisation sémantique. Cette approche consiste donc à associer aux lieux des informations; un ensemble des propriétés fonctionnelles d'un objet liées à son usage permettant à l'entité intelligente et de raisonner sur des concepts symboliques. Par exemple : interagir avec un humain de manière quasi-naturelle, classer les lieux en fonction des éléments qui les composent, traiter les ambiguïtés, détecter les erreurs de localisation. [5]

6.2. Représentation de l'environnement

C'est la discrétisation et la représentation de l'espace d'interaction virtuel par des formes géométrie statique simples telles que des carrés ou des segments, il permettre de traduit la structure de l'environnement par définir l'espace occupé par les obstacles ainsi que l'espace libre et donc les contraints imposé lors de la navigation. Cette discrétisation

permettre la représentation topologique de l'environnement à travers de la notion de l'accessibilité entre zone ce qui est nécessaire pour la recherche d'un chemin entre deux points. On distingue deux catégories pour la représentation de l'environnement la représentation approximative et la représentation exacte

6.2.1. Représentation approximative

Cette représentation est fortement utilisée en animation comportementale du fait de leur simplicité de mise en œuvre et leur rapidité d'exploitation. Il y a deux modèles entrant dans cette catégorie :

A. Décomposition à base de grille

Il y a deux types de grille selon la précision de la représentation :

- **Grille régulières**

C'est un modèle approximative utilise des grille uniformes régulières couvrent tous la surface. Donc l'environnement est pavé par des cellules, qui peuvent avoir trois états : libre, partiellement obstruée, et obstacle. La taille des cellules utilisés joue un rôle très important dans la précision de représentation. Au fur et à mesure que la taille des cellules diminue, la représentation est plus précise, ce qui conduit à une augmentation de l'utilisation de la mémoire .L'occupation mémoire de cette méthode constitue ainsi son premier point faible.

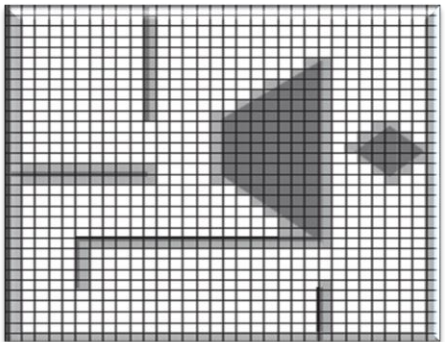


Figure 1.7. Grille régulières

- **Grille hiérarchiques**

Une évolution de ce modèle a été proposée sous forme de grilles décrire l'espace navigable par une succession de grilles de plus en plus précises, organisées sous forme d'arbre. Ce qui permet de réduire le nombre de cellules dans la grille en conservant de l'information détaillée là où il y en a besoin.

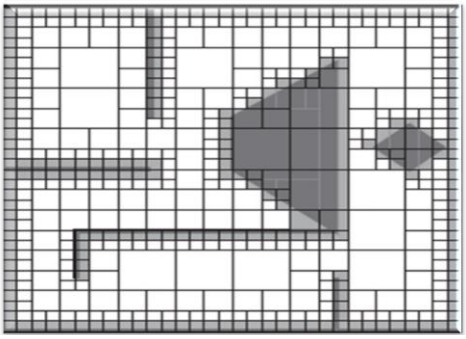


Figure 1.8. Grille hiérarchiques

L'utilisation d'un quadtree permet de réduire considérablement le nombre de cellules de la grille, et donc l'utilisation mémoire. Il est ainsi plus efficace de représenter de larges environnements ouverts en focalisant le raffinement sur les frontières entre C-Free et C-Obstacle. Cette structuration hiérarchique est faite à l'aide d'un quadtree structurant les données des cellules. [10]

B. Décomposition cylindrique

À l'aide d'une décomposition en un ensemble de cylindres l'environnement navigable est représenté. L'utilisation de ces cylindres lui permet de créer une structure comprenant un nombre assez restreint D'éléments et permettant la navigation de plusieurs milliers d'agents en temps réel.

Les axes les plus éloignés des obstacles et présentant donc la plus grande sécurité au niveau de la navigation, sont tout d'abord identifies. Des cylindres, centres sur ces axes et dont le rayon correspond à la distance de l'obstacle le plus proche, sont définis le long de ces axes. Ces cylindres représentent donc dessous espaces de C-Free ou la navigation est possible. En connectant ensuite les cylindres qui s'intersectés, des couloirs de navigation ou les entités peuvent naviguer librement sont créés. En identifiant les axes médians et l'éloignement maximum aux obstacles, cette méthode construit une représentation relativement proche d'un diagramme de Voronoi généralisé. [10]

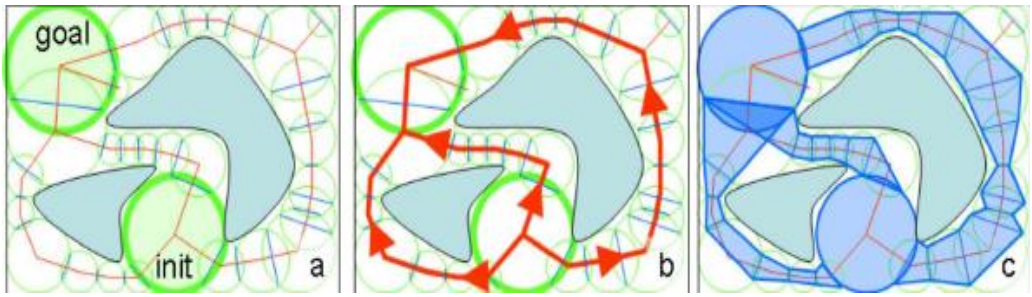


Figure 1.9. Représentation à l'aide des cylindres

C. Les cartes de cheminement

Cette approche consiste à calculer un réseau de chemins normalisés (lignes, courbes) passant à travers l'espace libre. Ce réseau est obtenu en reliant des points clefs répartis à l'intérieur de l'environnement ils existent plusieurs méthodes pour la création des cartes de cheminement, différentes dans la manière de créer et de relier ces points clefs. (Figure 1.10.).

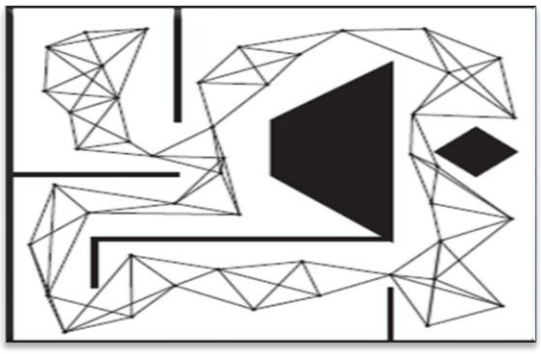


Figure 1.10. Carte de route probabiliste (PRM)

Des exemples de ces méthodes Cartes routières probabilistes, Visibilité, et Voronoi Diagarams.

Cette méthode présente l'avantage de fournir une description très condensée de l'environnement et produit des animations visuellement plausibles. Cependant, son fonctionnement est trop éloigné du comportement humain pour être exploité dans des simulations réalistes. [6]

C.1 Cartes de cheminement déterministe

- Graphe de visibilité

C'est une relation des éléments de cet ensemble vers les éléments de cet ensemble. La relation d'un élément vers un autre élément est définie si le second est visible à partir du premier suivant le modèle de visibilité qui a été établi.

Les nœuds du graphe de visibilité sont les sommets des enveloppes convexes approximant chaque région. Les arcs (non orientés) du graphe de visibilité relient les nœuds (couples d'´éléments) mutuellement visibles selon la d´définition qui vient d'être proposée.

La figure 1.11 représente plusieurs obstacles ainsi que le graphe de visibilité correspondant. En particulier, les zones hachurées représentent les obstacles et les arcs en pointillés représentent les segments de droite (forcément inclus dans l'ensemble des arcs du graphe de visibilité) constituant les obstacles. [7]

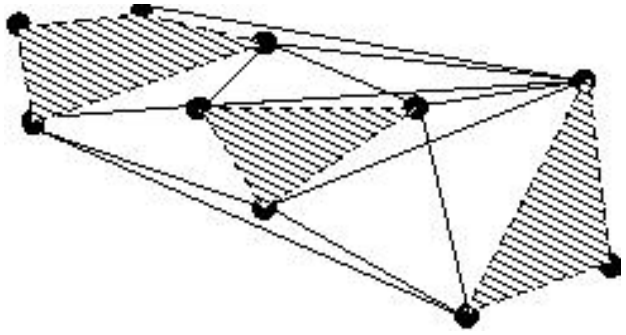


Figure.1.11. Graphe de visibilité d'un ensemble d'obstacles

Le réseau de chemins ainsi créé possède la propriété de maximiser la longueur des parcours en ligne droite, permettant ainsi de minimiser la distance parcourue [8]

Cependant, la taille du graphe généré est directement dépendante du nombre de paires de points mutuellement visibles. Considérons que dans l'environnement, ***n*** points soient mutuellement visibles. Chacun de ces points est relié aux ***n-1*** autres points. En considérant que le graphe est non orienté, ces ***n*** points gérèrent donc ***n(n-1)/2*** liens de visibilité. Un environnement ouvert, de grande taille, avec des obstacles ponctuels disparates, possédant donc un grand nombre de relations de visibilité, constitue l'un des pires cas pour la construction de ce type de graphe de cheminement.

- **Diagramme de Voronoï généralisé**

Cette méthode est basée sur une notion d'équidistance aux obstacles de l'environnement. Pour se faire, un ensemble de sites sont évalués au sein de l'environnement, correspondant aux obstacles, dont les intersections vont former les points clefs. Les chemins ainsi générés maximisent la distance aux obstacles, où le calcul est obtenu directement en effectuant le rendu des sites. Une autre méthode

utilise la triangulation de Delaunay pour obtenir les points clefs du diagramme de Voronoï généralisé, en se servant du centre des triangles calculés [9]. La carte de cheminement ainsi produite peut être considérée comme un condensé des informations de la triangulation, ne contenant plus la définition géométrique des obstacles de l'environnement.

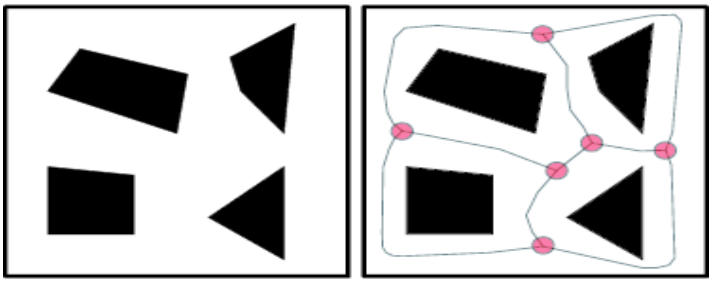


Figure.1.12. Graphe Voronoï généralisé. [9]

C.1 Cartes de cheminement probabilistes

Plutôt que de se baser directement sur les informations géométriques pour construire une carte de cheminement et identifier les points clefs, les cartes de cheminement probabilistes, s'appuient sur une génération aléatoire de points clefs pour couvrir l'espace libre. Deux points peuvent ensuite être reliés s'il existe, entre eux, un chemin libre de collision. Ce test constitue l'opération la plus coûteuse. Il exploite la géométrie de l'environnement et de l'entité en déplacement pour vérifier que le chemin généré ne provoque pas de collision.

Cette méthode est très utilisée dans le cadre de la planification de chemin pour des objets poly-articulés tels que des bras par exemple. Dans le cadre de la locomotion cette technique a permis la génération de mouvements très complexes adaptés à la difficulté de l'environnement, tels que des sauts, marcher sur des piliers, se baisser pour éviter une branche… .

Pour conclure, les cartes de cheminement présentent le net avantage de fournir une description très condensée de l'environnement, ne nécessitant une prise de décision qu'au niveau des points clefs. Néanmoins, leur définition seule n'est pas suffisante pour gérer la complexité de la locomotion humaine. Par exemple, leur exploitation devient très difficile lorsqu'il s'agit de gérer le croisement des entités le long d'un même chemin. Certaines méthodes proposent de régler ce problème en

subdivisant chaque chemin dans sa largeur, les gérant ainsi comme un ensemble de rails parallèles entre lesquels vont pouvoir transiter les entités en mouvement. Mais, même si une telle méthode produit des animations visuellement plausibles, son fonctionnement est trop éloigné du comportement humain pour être exploité dans des simulations réalistes. Les représentations sous forme de cartes de cheminement semblent donc bien adaptées à un processus de planification de chemin général,

Ne tenant pas compte des autres entités, mais beaucoup moins à un processus de navigation, gérant lui les mouvements locaux. [10]

6.2.2 Représentation exacte

C'est La décomposition spatiale exacte en cellules convexes de différentes formes vise à organiser les données spatiales afin de représenter l'environnement avec précision tout en préservant intégralement les informations qu'elles contiennent à l'origine ça veut dire ses caractéristiques géométriques et topologiques. Les techniques de décomposition la cellule exacte décomposent un environnement en une collection de cellules de sorte que l'union de toutes les cellules soit exactement égale à l'environnement. La géométrie des cellules peuvent être des triangles ou des polygones convexes [6].

A. Triangulation de Delaunay

Cette représentation reproduit **exactement** l'environnement d'origine tout en l'organisant de manière plus accessible. En utilisant les bordures des obstacles comme des contraintes formant les données d'entrées, pour caractériser précisément les espaces libres.

Les méthodes basées sur les triangulations de Delaunay contraintes utilisent ainsi les frontières des obstacles de l'environnement afin de définir la triangulation l'espace libre est donc représenté par un ensemble de triangles dont les arêtes libres représentent des connexions entre zones navigables et les arêtes contraintes des obstacles. La planification de chemin se ramène alors à identifier une séquence de triangles relies par des bordures franchissables. [11]

B. Décomposition en trapèzes

Cette décomposition permet de décomposé l'environnement sous forme de cellule trapézoïdales .elle est basé sur un algorithme de balayage. Les points délimitent la géométrie de l'environnement sont triées suivant l'axe des ordonnées. Puis un maximum de deux segment est généré, ayant pour origine le point sélectionné et pour extrémité la prochaine intersection avec le bord d'un polygone délimitant un obstacle (Figure1.13) [9]

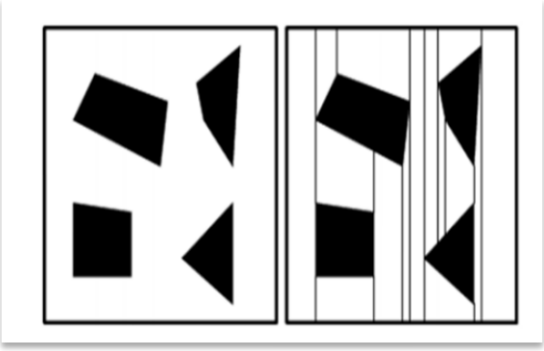


Figure 1.13. Décomposition en trapèzes

6.2.3 Le modèle de champs de potentiels

La méthode du champ potentiel repose sur un gradient de forces, ou champ de potentiel qui est déduit en chaque point de l'environnement comme étant une somme pondérée, le plus souvent par la distance, des potentiels de répulsion et du potentiel lié au but. Les champs potentiels l'approche se prête facilement aux extensions.

Par exemple, puisque les champs potentiels sont additifs, l'ajout d'un nouvel obstacle est facile car le champ correspondant à cet obstacle peut être simplement ajouté le à l'ancien. Cependant, les champs potentiels ont certaines limites, en particulier leur calcul complexité.(Figure 1.14.)

De plus, l'inconvénient majeur de la méthode est l'existence de minima locaux. Parce que l'approche de champ potentiel est une méthode locale plutôt que globale (elle ne considère le meilleur plan d'action immédiat), l'entité en mouvement peut rester coincée dans un minimum local de le champ potentiel fonctionne plutôt que de se diriger vers le minimum global, qui est la destination cible. [6]

Des champs de potentiel dynamiques ont été utilisés pour intégrer la navigation globale avec des obstacles et des personnes en mouvement, résolvant efficacement le mouvement de grandes foules sans qu'il soit nécessaire d'éviter explicitement les collisions [12]

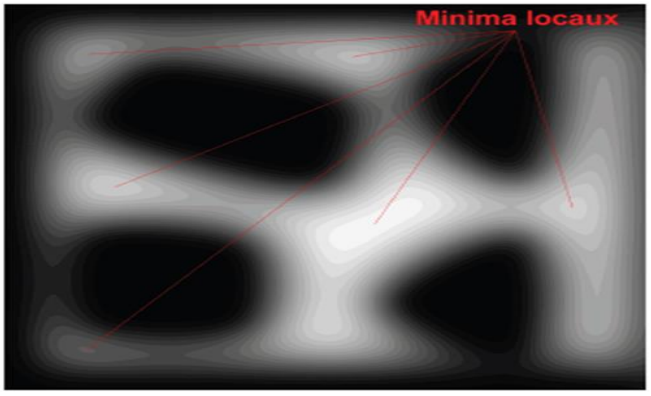


Figure 1.14. Le modèle de champs de potentiels

7. Représentation de l'Agent virtuel

La définition du mot *agent* qui vient du latin « *agere* » qui signifie agir littéralement, l'agent est donc celui qui agit ce qui nous produit des *agents* diverses Les différences relèvent du domaine d'application et du degré de complexité. Une définition plus proche de notre domaine est de Jacques Ferber [13] qui à définit un agent comme étant une entité physique ou virtuelle évoluant dans un environnement dont il n'a qu'une représentation partielle et sur lequel il peut agir. Il est capable de communiquer avec d'autres agents et est doté d'un comportement autonome ; ou L'autonomie est la faculté d'avoir ou non le contrôle de son comportement sans l'intervention d'autres agents ou d'êtres humains.

Un agent est une entité qui a comme tâche d'accomplir un ensemble d'objectifs dans un environnement dynamique et complexe.

7.1 Les différents types d'agents

On distingue trois types d'agent virtuel selon leurs comportements dans l'environnement virtuel :

7.1.1 Les agents réactifs

Les agents réactifs sont connues par des comportements stimuli-action, c'est-à-dire, ses actions sont choisies et exécutées selon les événements perçus dans l'environnement.

Il déclenchera une action prédéfinie pour chaque configuration de ses capteurs, sans passer par la phase de planification à partir d'une représentation interne de son environnement.

7.1.2 Les agents Cognitifs

Un agent cognitif (aussi appelé agent délibératif) une entité utilise une représentation interne de son environnement afin de formuler des plans d'actions pour satisfaire ses buts. Il est plus autonome et possède un niveau de connaissance plus élevé.

7.1.3 Les agents hybrides

Pour obtenir une architecture capable d'exhiber à la fois des comportements réactifs et cognitifs, les chercheurs ont proposé un nouveau type d'architecture appelée l'architecture hybride.

Les agents hybrides sont généralement des agents cognitifs intégrant des capacités réactives. L'association des deux méthodes permet de combiner leurs qualités complémentaires (planification, abstraction, réactivité) tout en limitant leurs défauts respectifs [14].

7.2 Les Comportements d'agent virtuel

7.2.1. Comportement individuel

Les comportements individuels sont proposés par **Craig Reynolds** ces comportements simple sont réalisé par chaque entité on utilisant l'information locale afin de simuler ce dernier, ces comportements sont réalisé par des forces sociale qui vont aider l'agent autonome à se déplacer d'une manière cohérente [9].

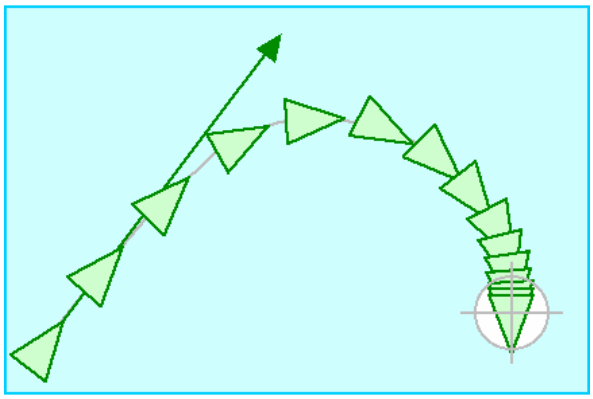
Un individu est capable d'élaborer un plan comportemental lui permettant d'ordonner ses actions, et de gérer leurs liens de dépendance. Ce raisonnement va ainsi avoir un impact direct sur son déplacement dans l'environnement. Parmi ces comportements en site les comportements suivants :

7.2.1.1 Comportement d'arrivée

Le comportement d'arrivée est une extension du comportement de recherche. De même que pour le comportement de recherche, il est utilisé pour orienter

l'individu vers une cible indiquée. La différence importante se résume dans la manière selon laquelle l'individu atteint la destination.

Le comportement de recherche fait que notre individu atteint la cible à pleine vitesse. Il se déplace en fait davantage dans la direction actuelle et va ainsi générer plutôt un mouvement de danse semblable au comportement d'une mite autour d'une source lumineuse. Cependant, le comportement d'arrivée doit être tel qu'il y'ait un ralentissement commandé duvéhicule conformément au cahier des charges édicté par l'utilisateur, le mouvement doit s'arrêter à la position désirée (Figure 1.15). Ceci fait provoque le ralentissement du véhicule selon sa distance actuelle à la destination et



le résultat est un véhicule qui s'arrête au niveaude la cible indiquée [15]

Figure 1.15. Le comportement d'arrivée.

7.2.1.2 Comportement d'évitement d'obstacles

Le comportement d'évitement d'obstacles donne à un acteur la capacité de manœuvrer dans un environnement encombré en esquivant autour des obstacles. Il y a une distinction importante entre l'évitement d'obstacles et le comportement de fuite.

La fuite a pour conséquence d'orienter l'acteur loin d'un emplacement donné, tandis que l'action d'évitement d'obstacles agit seulement quand un obstacle se trouve directement devant cet acteur. Par exemple, si un individu se déplace sur une trajectoire parallèle à un mur, l'évitement d'obstacles ne prendrait aucune mesure corrective de direction, mais la fuite essaierait de s'éloigner du mur, en suivant la perpendiculaire à ce mur.

Le but du comportement est de garder un cylindre imaginaire de l'espace libre devant l'acteur. Le cylindre se trouve le long de l'axe vers l'avant l'acteur, à un

diamètre égal à la sphère de bondissement de l'acteur, et s'étend du centre de l'acteur pour une distance basée sur la vitesse et l'agilité de l'acteur.

Le comportement d'évitement d'obstacles considère chaque obstacle alternativement et détermine si ces obstacles possèdent des intersections avec le cylindre. En localisant le centre de chaque obstacle sphérique, le test d'intersection avec le cylindre est ainsi effectué très rapidement [15]

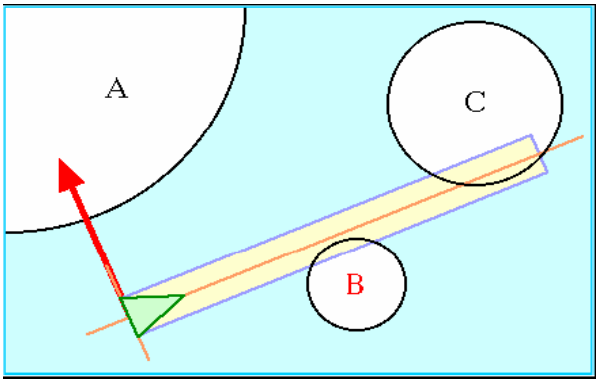


Figure 1.16 L'évitement d'obstacle.

Une note finale concernant l'interaction d'évitement d'obstacles et de rechercher de but. D'une façon générale nous nous intéressons seulement aux obstacles qui sont localisés entre l'acteur et le but à atteindre.

7.2.1.3 Comportement de suivi de chemin

Ce type de comportement permet à un acteur de s'orienter le long d'une voie d'accès prédéterminée, telle qu'une chaussée, un couloir ou un tunnel. C'est en fait très distinct de l'action de contraindre un individu à suivre d'une manière rigide une voie d'accès tel que le roulement d'un train le long d'un rail, par exemple. Le comportement de suivi d'une voie d'accès est plutôt destiné à la production de mouvements tels que des personnes empruntant un couloir : les différentes voies d'accès restent proches et sont souvent parallèles à l'axe du couloir, elles sont néanmoins libres [15]

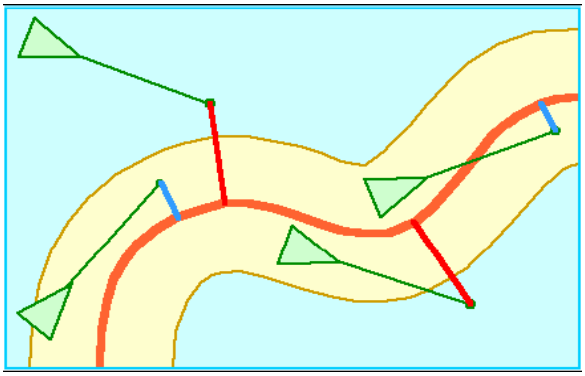


Figure 1.17. Le comportement de suivre de chemin.

Autrement dit, l'acteur vire loin de la voie d'accès, ou en est trop loin de celle-ci. Pour l'orienter en arrière vers la voie d'accès, le comportement de recherche est exploité pour une orientation en arrière vers la projection de la voie d'accès de la future position prévue, de la même manière que pour l'action d'évitement [16] .

7.2.1.4 Evitement de collision non-alignée

Le comportement d'évitement de collision non alignée se doit de prévenir les collisions entre des acteurs se déplaçant dans une direction arbitraire. Considérez votre propre expérience de marche à pied à travers une place ou une entrée pleine d'autres marchants : L'évitement de collisions implique la prévision de collisions potentielles et le changement de votre direction et de votre vitesse pour les prévenir (Figure 1.18) [15] .

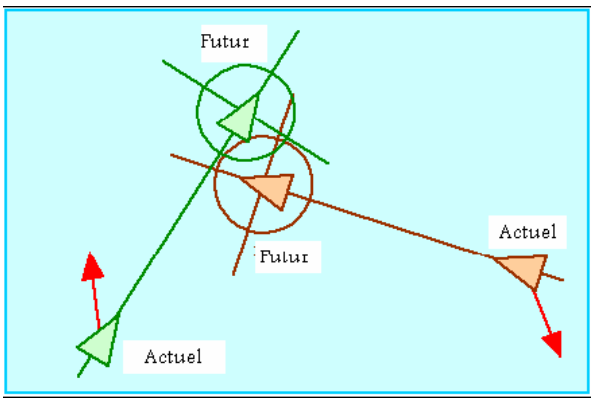


Figure 1.18. Le comportement d'évitement de collisions non-alignées.

7.2.2. Comportement de groupe

Dans son approche de simulation de comportements de groupes, Reynolds simule les nuées d'oiseaux, les troupeaux d'animaux et les bancs de poissons [17] . Un groupe est l'élaboration d'un système de particules avec acteurs (oiseaux ou poissons) en tant queparticules.

Un groupe est supposé le résultat de l'interaction entre les comportements individuels des acteurs. De façon individuelle, les acteurs tentent de rester ensemble et d'éviter les collisions avec les autres acteurs, d'une part et les objets aux alentours, d'autre part. Les positions, les vitesses et les différentes orientations des acteurs sont connues dans le système à tout instant. L'animateur peut contrôler plusieurs paramètres globaux: [14]

- Le poids du composant évitement d'obstacles.
- Le poids de la convergence vers le but.
- Le point du centrage du groupe.
- Le poids de l'égalité des vitesses.
- La vitesse maximale.
- L'accélération maximale.
- La distance minimale entre les oiseaux.

Les trois comportements suivants de direction (séparation, cohésion et l'alignement) touchent des groupes d'acteurs. Dans chaque cas, le comportement de direction détermine comment un acteur réagit à d'autres dans son voisinage local. Les acteurs à l'extérieur du voisinage local sont ignorés. Le voisinage est spécifié par une distance qui est définit quand deux acteurs sont "voisin" et un angle qui définit la perception de l'acteur "champ de voisin" (Figure 1.19).

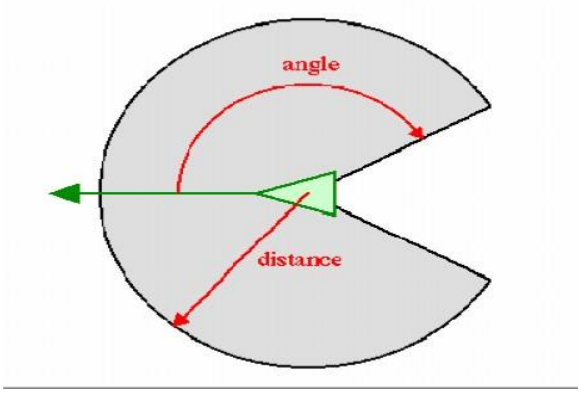


Figure 1.19. Notion de voisinage.

7.2.2.1 La séparation

Le comportement de séparation donne à l'acteur la capacité de maintenir une certaine distance de séparation des autres caractères qui lui sont proches. Cela peut être employé pour empêcher les acteurs de s'entasser ensemble. Pour calculer la direction de séparation, d'abord une recherche est faite pour trouver d'autres acteurs dans le voisinage indiqué. Pour chaque acteur voisin, la force répulsive est calculée en soustrayant les positions de notre acteur et l'acteur voisin, ensuite la normalisation de l'application à **1/r** indemnité. (C'est-à-dire la position compense le vecteur est mesuré par **1/r2**). Notons que **1/r** est juste un arrangement et non une valeur fondamentale [15] .

Les forces répulsives pour chaque acteur voisin sont additionnées ensemble pour produire la force de direction (Figure 1.20).

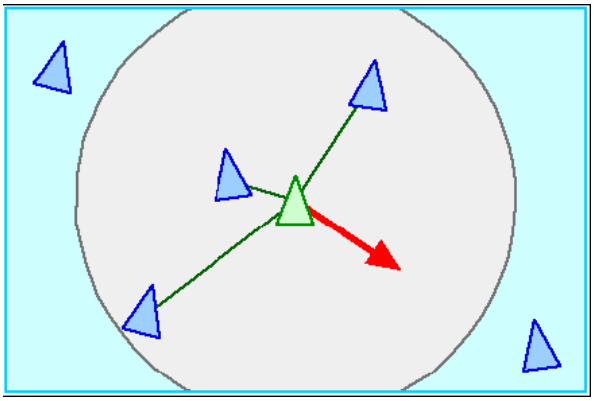


Figure 1.20. Le comportement de séparation. [15]

7.2.2.2 La cohésion

Le comportement de cohésion donne aux acteurs la capacité de générer une situation de cohérence dans le groupe qu'ils forment (se rapprocher et former un groupe) avec d'autres acteurs voisins (Figure 1.21).

La direction de la cohésion peut être calculée en déterminant tous les acteurs dans le voisinage local (comme décrit plus haut pour la séparation) et en calculant "la position moyenne" (ou "centre de gravité") des acteurs voisins. La direction de la force peut ainsi être appliquée dans la direction de "la position moyenne" [15].

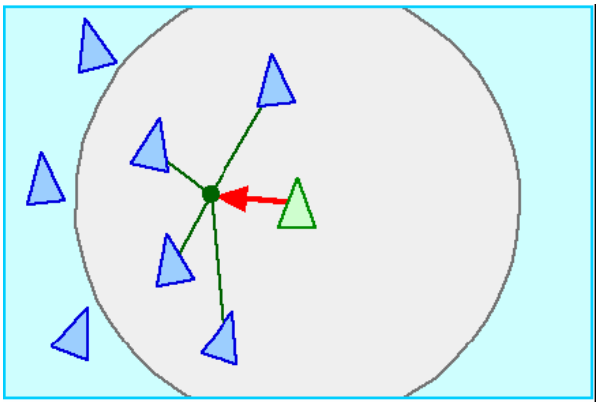


Figure 1.21. Le comportement de cohésion.

7.2.2.3 L'alignement

Le comportement d'alignement donne à l'acteur la capacité de s'aligner sur d'autres acteurs voisins (Figure 1.22).

La direction d'alignement peut être calculée en déterminant tous les acteurs dans le voisinage local (comme décrit précédemment pour la séparation), en faisant la moyenne des ensembles des vitesses des acteurs voisins.

Cette moyenne est "la vitesse désirable", ainsi le vecteur de direction est la différence entre la moyenne et la vitesse courante de notre acteur. Cette direction a tendance à faire tourner notre acteur, il est ainsi aligné sur ses voisions [15].

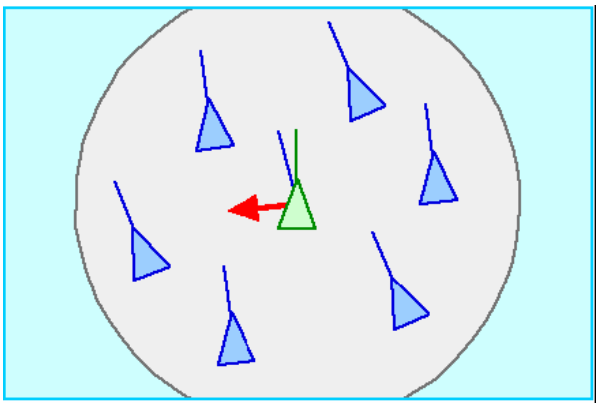


Figure 1.22. Le comportement de d'alignement. [15]

Additivement aux comportements de séparation, de cohésion et le comportement d'alignement, ces comportements peuvent être combinés pour produire le modèle de comportement de masse, de troupeau et ou de colonie. En général, et pour la majorité des applications, il suffit simplement de résumer les trois directions des vecteurs de force pour produire une direction simple combinée pour les masses. Cependant, et pour le meilleur contrôle, il est utile d'abord de normaliser les trois composantes de direction pour les mettre ensuite à l'échelle par trois facteurs d'indemnité avant dès les récapituler.

7.2.2.4 Le suivi de leader

Le comportement de suivi de leader, est provoqué par un ou plusieurs acteurs à l'effet de suivre un autre acteur en mouvement et considéré comme le leader. Généralement, ces acteurs tentent des rester près du leader, sans la cohue et faisant attention à rester loin de la voie du leader (dans le cas où ils arrivent à trouver des disciples devant le leader). De plus,s'il y a plus d'un disciple, ils doivent éviter de le heurter (Figure 1.23) [15] .

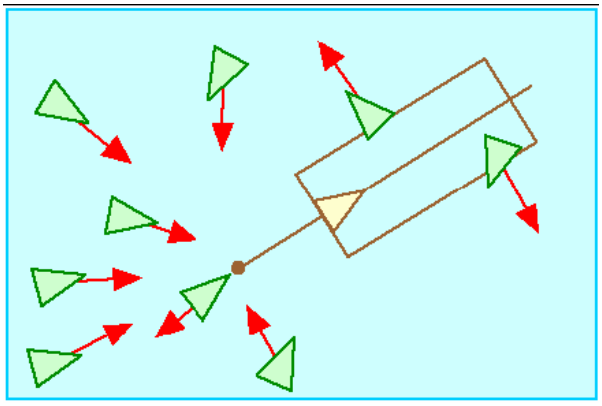


Figure .1.23. Comportement de suivi de leader. [15]

8. Conclusion

On conclut que l'animation comportementale est un domaine multidisciplinaire comprend le domaine de l'informatique et le domaine de socio-psychologie ou l'étude des caractéristiques psychologique et des forces sociologues affecté sur le comportement de l'individu pendent l'interaction avec son environnement constitué d'un ensemble d'obstacle statique et dynamique, ce qui nous conduisons vers la représentation topologique de l'environnement traduit son structure et définit le champ de navigation

On a vu aussi l'individu ou appeler l'agent virtuel et ces défirent types selon son comportement dans l'environnement virtuel, ces comportement peut être individuel ou collectifs qui est le résultat de l'interaction entre les comportements individuels des acteurs.

Dans le chapitre suivant on va étudier La simulation comportementale qui représente la deuxième partie de l'animation comportementale.

Chapitre 02:

La Simulation comportementale

1. Introduction

Dans ce chapitre, on s'intéresse à la simulation comportementale, la modélisation du comportement individuel ou chaque individu à son propre comportement et un comportement de groupes ou les individus sont en interaction entre eux. Nous présentons les déférentes approches et les modèles de simulation proposée par les chercheurs et leurs caractéristiques et une brève comparaissent entre ces modèles. Ainsi que les algorithmes de planification de chemin pour la navigation des individus dans l'environnement

2. Les Méthodes de simulation comportementale

La simulation comportementale est une partie de l'animation qui se rapproche des systèmes réels de par son principe de fonctionnement en assignant aux acteurs ou systèmes animés des comportements indépendants. Trois approches de simulations sont définies pour la simulation de la dynamique de populations composées d'individus en interaction: l'approche microscopique, l'approche macroscopique et l'approche misoscopique.

2.1 Modèles microscopique

L'approche microscopique permet des simulations dans lesquelles chaque entité est gérée individuellement. Un individu aura donc ici son fonctionnement propre qui prend en compte son environnement dont les entités (agents ou objets) [18]

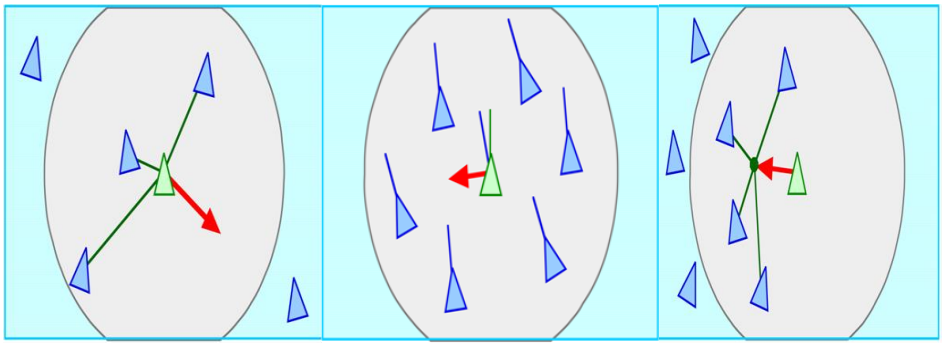
Dans les modèles microscopiques, le mouvement de chaque individu est représenté dans le temps et l'espace. Chaque individu a son propre comportement, ses propres décisions et interagit avec les autres. Ces modèles sont classifie en 3 approches: les modèles à base de règles, les modèles de forces sociales, et les modèles d'automates cellulaires. [19]

2.2.1Modèle à base de règles

Craig Reynolds [15], avec son modèle Flocks of Boids, introduit la notion de simulation microscopique à base de règles. Ici, le déplacement de chaque individu

est régit par des règles de comportement de la forme « si condition alors action ».Trois règles sont proposées par C. Reynolds dans le cadre de l'animation de nuées (Figure 2.1.) :

- ✓ Séparation afin d'éviter d'éventuelles collision avec ses voisins.
- ✓ Alignement afin de réguler sa vitesse par rapport à l'ensemble du groupe.
- ✓ Cohésion afin de rester proche de ses voisins.



a) Séparation b) Cohésion (c) Alignement

Figure 2.1. Règles locales de Reynolds.

Pour conclure, les modèles à base de règles proposent une approche plus souple de la navigation microscopique, permettant d'introduire des adaptations variées dans différentes situations. Néanmoins, le caractère itératif de la décision rend difficile la fusion d'informations, les règles étant indépendantes les unes des autres.

2.1.2 Modèles de forces sociales

C'est une approche microscopique pour simuler le mouvement piétonnier développé par D. Helbing [4]. Ce modèle est basé sur un système de forces socio-psychologiques traduisant deux caractéristiques: premièrement la tendance à garder une certaine distance avec les autres piétons, deuxièmement la friction intervenant lors de la résolution d'un contact entre plusieurs piétons. La force sociale donnée telle que l'interaction répulsive, force de frottement, dissipation est analogue à de vraie force, il résout les équations du mouvement de Newton pour chaque individu. Chaque agent est représenté dans la locomotion plane par un cercle avec son propre diamètre et le modèle décrit les coordonnées, les vitesses, et les interactions continues avec d'autres agents, chaque paramètre de force a une interprétation réelle,

et il est différent pour chaque piéton. Les forces sociales modèlent le comportement de foule humain avec un mélange de facteurs socio-psychologique et physiques.

Ce modèle peut être appliqué avec succès pour simuler les scénarios de mouvement piétonniers réels principalement dans la gestion du comportement de pétions dans des situations d'évacuation. Il est caractériser par l'avantage de faible complexité, ainsi que le réalisme car il n'y a pas de discrétisation spatiale de la zone de déplacement et les valeurs des forces sont mesurées de manière précise. Mais il n'est pas adapté aux foules trop denses car les piétons oscillent sur place de part l'utilisation de la force de répulsion à distance (Figure 2.2.) [20].

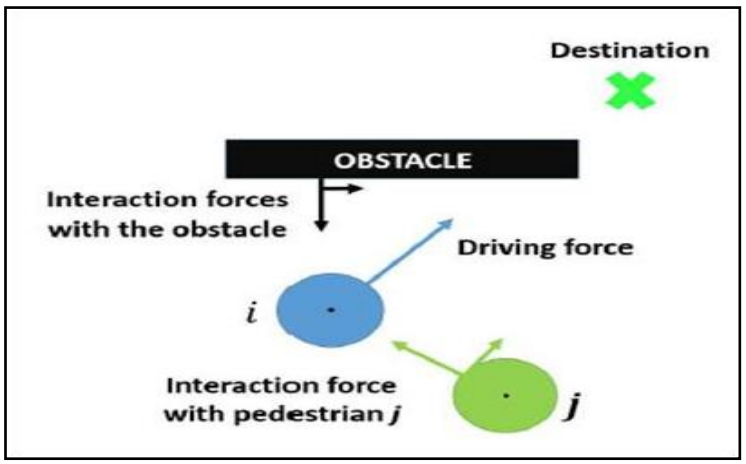


Figure 2.2. Les forces agissant sur le piéton *i*

3.2 Modèles d'automates cellulaires

Les modèles d'automates cellulaires sont des modèles où l'espace de déplacement 2D des piétons est discrétisée [21]. Une grille uniforme de cellules comporte des cellules inaccessibles pour représenter les obstacles, des cellules occupées par les piétons et des cellules vides. Leur principe repose sur le fait que chaque piéton occupe une cellule et ne peut se déplacer vers une autre cellule que si celle-ci est libre. A chaque pas de temps, le piéton a une direction de préférence qui permet de construire une matrice 3*3 (représentant la cellule courante et chaque cellule adjacente, par le coin ou l'arête) contenant les probabilités de déplacement du piéton (Figures 2.3). Cette matrice permet de déterminer la prochaine cellule sur laquelle le piéton va se déplacer. Si celle-ci est occupée, le piéton va rester sur sa cellule.

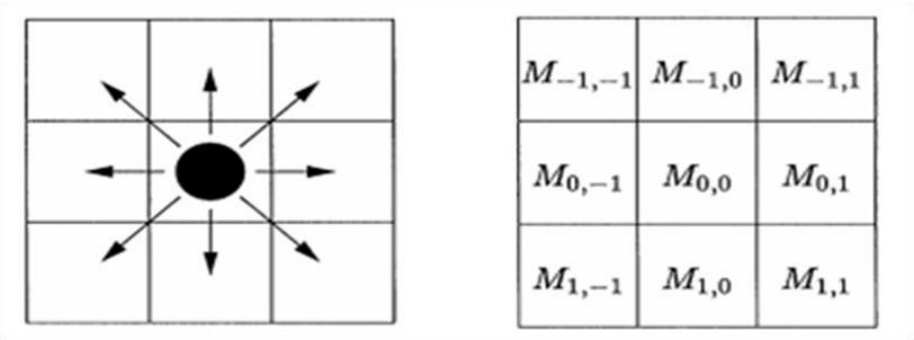


Figure 2.3. Automates cellulaires Matrice 3*3 des probabilités de la prochaineposition d'un piéton.

Le principal avantage de ce type de modèles est la réalisation de simulations dans des espaces vastes avec beaucoup de piétons car les temps de calcul des simulations sont intéressants. L'inconvénient majeur est le manque de réalisme car le mouvement des piétons est restreint par la grille et les contacts ne sont pas gérés directement [18] .

2.2 Modèles macroscopiques

Les modèles macroscopiques consistent à traiter, agréger les mouvements des piétons et à s'intéresser à simuler des flux de déplacement des foules de piétons avec un très faible coût calculatoire et sont capables d'intégrer facilement des observations réelles. L'agent dans ces modèles n'est pas traité individuellement mais en tant qu'élément constitutif de la foule. Les modèles macroscopiques sont peu adaptables à de nouvelles situations et pour qu'elles fonctionner correctement Ils ont besoin d'informations issues d'observations réelles.

Il existe deux modèles importants :

2.2.1 Le modèle gazeux

En 1971 Henderson propose un modèle gazeux permettant de simuler des foules de piétons dans des conditions de faible densité à ceux des molécules de gaz. S'appuyant sur la théorie cinétique des gaz de Maxwell Boltzmann, Henderson établit un parallèle entre la densité des piétons et celle des particules d'un gaz, en assimilant leur différence d'états (arrêt, marche, course) aux différences d'énergie des gaz. [6] Ces modèles sont basés sur la dynamique des fluides est connue dans la modélisation du trafic de véhicules.

2.2.1Le modèle hydraulique ou modèles de flots

Pour la simulation des foules de forte densité un modèle hydraulique a été proposé par J.Archea [22] Il assimile ici le mouvement des personnes au travers de couloirs, escaliers, et portes, à celui de l'eau au travers de tuyaux, vannes, ou autres. Ce modèle est utiliser spécifiquement pour La simulation de l'évacuation du bâtiment qui se fait en deux étapes:

1. Initialisation du mouvement de l'ensemble des individus recevant simultanément Un message d'alerte (ouverture des vannes);
2. Simulation de l'évacuation (écoulement de l'eau dû à la force de gravitation). [10]

et dans le cadre de la simulation de fluides, les agents son simules à partir d'un modèle qui s'appuie sur les quatre hypothèses suivantes [23] :

1- chaque personne tente d'atteindre un objectif géographique.
2- les personnes cherchent à marcher à la vitesse la plus élevée possible.
3- il existe des endroits dits inconfortables que les piétons vont chercher à éviter.
4- le piéton cherchera à emprunter le chemin le moins " couteux ".

2. Modèles misoscopiques

Dans les modèles précédentes le premier modèle à traité le réalisme et tenir compte finement des caractéristiques individuelles avec un cout très important et le deuxième à une simulation avec le moindre coût mais négligent l'autonomie des piétons Plusieurs travaux proposent des approches hybrides afin de composer les deux niveaux de représentation dans un même modèle.

Yersin et al. (2008) propose la notion de ROI Region Of Interest pour mettre en place des simulations de déplacements d'une grande quantité de piétons permettant de combiner les avantages des deux types de représentation. Ainsi, les régions ayant un grand intérêt pour l'observateur, utilisent des modèles microscopiques, et les régions d'un intérêt moindre (par exemple : éloignées de la caméra) utilisent des modèles macroscopiques moins coûteux ressources de calcul. Le concept de ROI permet de fluidifier la simulation et d'améliorer le passage à l'échelle. Cependant, il pose également de nouvelles problématiques comme la distribution des régions sur l'espace de navigation, et la cohérence des déplacements aux frontières entre les différentes régions.

Anh et al. (2012) proposent un modèle hybride pour la simulation de l'évacuation de plusieurs piétons dans un espace urbain ayant la structure d'un réseau filaire.

Loscos et al. (2003) proposent de stocker dynamiquement les informations concernant la fréquentation et les directions des agents dans une grille 2D. Ces informations sont ensuite utilisées par les agents pour s'orienter. Karamouzas et al. (2009) présentent une approche similaire qui exploite un algorithme A* (Nilsson, 1980) pour permettre aux piétons de suivre des itinéraires qui contournent les zones denses. Les données sur la fréquentation de l'espace évoluent en fonction du déplacement des piétons et en fonction du temps.

Sud et al. (2008) exploitent un graphe de navigation dynamique dont la structure topologique est modifiée en temps réel et utilisent les liens pour accélérer les calculs d'évitement de collision et détecter les zones de congestion. Saboia et Goldenstein (2012) modifient le modèle de Helbing et Molnar (1998) et associent à chaque agent une grille de perception mobile qui lui permet de calculer des vitesses désirées en temps raisonnable et en tenant compte de la densité.

Ces travaux proposent plusieurs solutions techniques intéressantes pour la prise en compte de la congestion, mais n'insistent pas assez sur le cadre formel permettant d'appréhender la complexité du comportement, notamment, l'articulation entre la planification d'itinéraire et la prise en compte des interactions entre agent , et la nature des influences entre les agents, qui donne leur véritable sens aux informations stockées dans les infrastructures de navigation et qui justifie les méthodes de planification employées. L'importance donnée aux performances computationnelles semble prendre le dessus sur la recherche de la crédibilité proprement dite. [24]

3. Comparaison entre les approches de simulation

Les différences entre les approches de simulation sont résumées dans le tableau (1,1)

L'approche microscopique	L'approche macroscopique	L'approche misoscopique
Simulation dans lesquelles chaque entité est gérée individuellement. Chaque individu aura Son propre fonctionnement. Consommation élevée en termes de ressources de calcul.	Simulé des grand nombre de piétons. Ne se soucient pas les comportements individuels. Tous les piétons ont le même but et motivations. Faible consommation en terme de ressources de calcul.	Combinent les deux modèles Simules des entités individuelles et des groupes d'entités

4. Les Algorithmes de parcours de chemin

Le parcoure d'un chemin ce fait par des algorithmes qui permet à un personnage, objet ou une unité de se déplacer dans un environnement en partant d'un point de départ et en se dirigeant vers un point d'arriver ou destination tout en évitant les obstacles présents dans cet environnement. L'objet doit rester dans une configuration valide durant toute la durée du trajet. Dans la section suivante on va discuter à propos ces algorithmes commencent par l'algorithme de Dijkstra.

4.1 Dijkstra Algorithm:

L'algorithme de Dijkstra a été conçu par l'informaticien néerlandais Edsger Dijkstra en 1959 [25]. Cet algorithme opère directement sur le graphe topologique, qu'il soit explicite (par exemple avec les cartes de cheminement) ou implicite (par exemple avec les grilles). L'algorithme met à jour une table des poids estimés des plus courts chemins entre chaque sommet et la position courante de l'entité représenté par le sommet de départ et un compteur de distance nul.

Les sommets de la frontière de notre parcours de graphe (Figure 2.4), sont ici rangés dans une file de priorité, où il stocke pour chaque nœud exploré deux informations :

- la distance parcourue depuis le nœud de départ jusqu'au nœud courant ;
- le nœud précédent, i.e. parcouru avant le nœud courant.

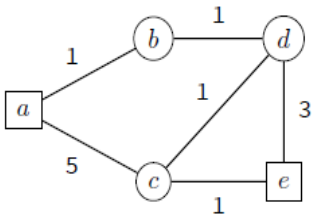
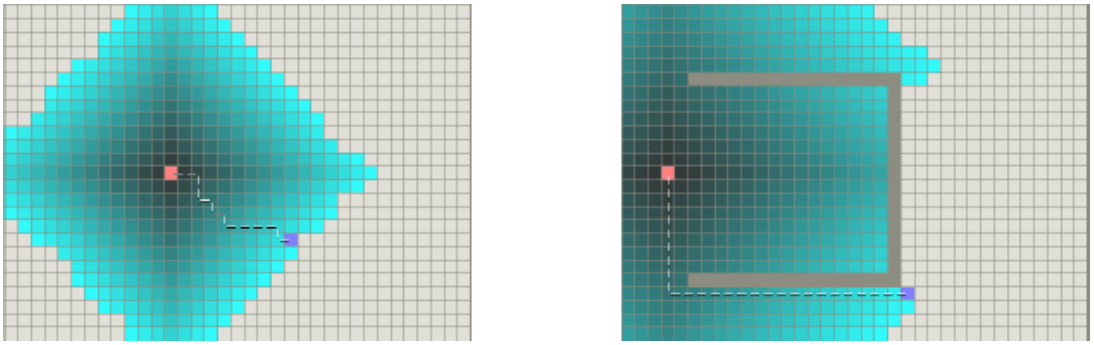


Figure 2.4 un petit graphe illustrant l'algorithme de Dijkstra

a	b	c	d	e	file de priorité
0	∞	∞	∞	∞	$\underline{a}$
0	1	5	∞	∞	$\underline{b}\,c$
0	1	5	2	∞	$c\,\underline{d}$
0	1	3	2	5	$\underline{c}\,e$
0	1	3	2	4	$\underline{e}$

Tableau 2.2 les étapes successives

Afin de trouver un chemin entre deux nœuds, en admettant que la longueur de ce chemin soit l, l'algorithme explorera par propagation circulaire l'ensemble des nœuds se trouvant à une distance inférieure à l. la figure 2.5 représente le fonctionnement de l'algorithme de Dijkstra dans des cas contraints ou non. Le carré rose représente le nœud source, le mauve la destination. Le gradient de bleus correspond à la distance à l'origine, le plus clair étant le plus éloigné. [26]



(a) Environnement non contraint (b) Environnement avec obstacle concave

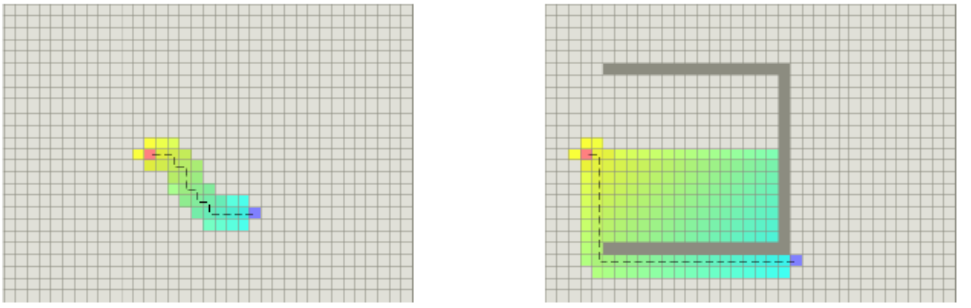
Figure 2.5. Fonctionnement de l'algorithme de Dijkstra.

L'algorithme de Dijkstra garantit de trouver le chemin le plus court du point de départ au but, tant qu'aucun des bords n'a un coût négatif. L'intérêt majeur de cet algorithme est sa robustesse (si un chemin existe il sera forcément trouvé).

Son point faible est son calcul étant de l'ordre de O (A + N. logN) où A est le nombre d'arcs du graphe, et N est le nombre de nœuds. [27]

4.2 A* Algorithme:

A-Star ou Search (A*) proposé pour la première fois par **Peter E. Hart , Nils John Nilsson et Bertram Raphael** en (1968). C'est un des algorithmes classiques les plus robustes car il trouve toujours un chemin s'il en existe un. Il allie également un bon ratio performance/qualité du chemin trouvé comparativement aux autres algorithmes. Il a été créé pour que la première solution trouvée soit l'une des meilleures, c'est pourquoi il est célèbre dans des applications comme les jeux vidéo privilégiant la vitesse de calcul sur l'exactitude des résultats, il fonctionne de la même manière que l'Algorithme de Dijkstra, mais il diffère dans son utilisation d'un heuristique pour chercher à se diriger vers le nœud d'arrivée, Cette heuristique doit satisfaire la contrainte de ne jamais surestimer la distance réelle.



(a) Environnement non contraint (b) Environnement avec obstacle concave

Figure 2.6. Fonctionnement de l'algorithme A* [26]

L'algorithme de A* se caractérise par sa vitesse de calcul Il ne donne pas toujours la solution optimale mais il donne très rapidement une bonne solution malgré cela l'utilisation d'une heuristique reste un majeur inconvénient. En effet, plus l'évaluation du chemin de longueur sera complexe, impliquant éventuellement d'autres paramètres que la distance (par exemple un coût associé à la sémantique), et plus cette heuristique sera difficile à expliquer .Ainsi, il est difficile d'utiliser cet algorithme pour la planification de chemin dont le but n'est pas clairement identifié dans le graphe topologique.

A* utilise deux listes de d'états : une liste «ouverte » et une «fermée». La liste ouverte contient toutes les cases où le personnage peut éventuellement se déplacer, alors que la liste fermée contient les cases déjà parcourues. L'heuristique évalue la distance jusqu'à l'arrivée à chaque case qu'elle traverse, généralement notée h, elle peut diminuer à chaque pas si elle s'approche du point de destination par rapport au précédent pas. Une autre variable est g, mesurant le chemin parcouru. Ainsi plus le nombre de déplacements est élevé, plus la variable est importante. L'algorithme additionne les variables g et h pour donner la variable f Toutes les variables f de chacun des chemins pris par l'algorithme sont ensuite comparées afin de trouver le chemin le plus court. L'algorithme s'arrête dès qu'il a trouvé le point d'arrivée et retourne le chemin parcouru pour s'y rendre. [28]

4.3. Evolution de l'algorithmes A*

Comme A* est un algorithme datant de 1968, il à connus de plusieurs variantes et de nombreuses évolutions, durant le temps, une inspiration de dynamique de A* appeler D* il prend en compte le fait que les chemins puissent changer de coût pendant que l'algorithme s' exécute. [28] .En 1996 Kavraki et al., on inventé une méthode **PRM** qui utilisé surtout dans la robotique, utilise A*comme base, l'algorithme se base sur les espaces libres de l'environnement. Il teste chaque échantillon aux alentours du robot pour voir si c'est un chemin possible. Ensuite un algorithme de recherche de graphe est utilisé sur les résultats pour trouver un chemin. Apré Sturtevant et Buro,en 2005 on crées un algorithme **(PRA*) Partial Refinement A*** qui produit plusieurs niveaux d'abstraction entre les cases d'un quadrillage afin de trouver un chemin entre le départ et la estination plus rapidement. Le démarche de l'evolution de l'algorithme A* se continué par la dévelopement de l'algorithme **(IDA*) lterative-Deepening A-Star Search** son qvantage est stoppé lorsque que le coût du chemin calculé par A* dépasse un seuil précisé au lieu de stopper le parcours des chemins avec une profondeur seuil. Il y a aussi l'algorithme **(HP A*)Hierarchical Pathfinding A*** Cet algorithme (Botea, Müller et Schaeffer, 2004) divise une carte de jeu en plusieurs sections de façon à simplifier les déplacements entre les grandes zones, chemins prédéfinis et déjà calculés, La figure 2.7 montre la division de la carte en de nombreux points. Les chemins entre ces points sont donc précalculés. **(gHPA *)Generalized Hierarchical Pathfinding A*** qui est une variante de HP A* (Koch, 2011) permet de mieux prendre en compte les graphes orientés lorsque ceux-ci n' ont pas des arêtes avec un poids négatif par rapport au HPA * standard. [28]

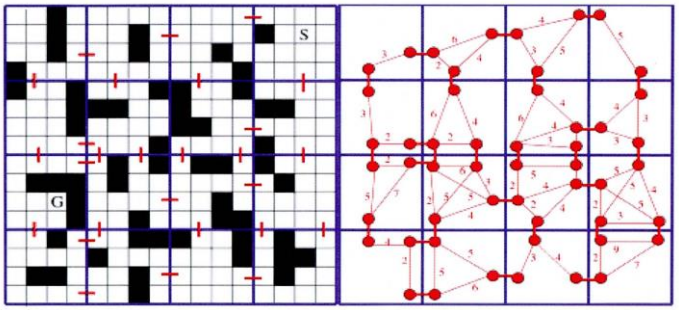


Figure 2.7. Hierarchical Path Finding (Botea, Müller et Schaeffer, 2004) [28]

5. Conclusion

En concluant ce chapitre, la simulation comportementale exprime la relation entre l'agent virtuel et son environnement virtuel , pour cela on a étudier les approches de simulation comportementale de l'agent virtuel et les différents modèles de simulation faites pour des individus en interactions, pendent leur déplacement en évitant les collisions et les obstacles d'une façon plus réaliste pour cela il faut introduire les algorithmes de planification de chemin, comme Dijkstra et A* ou bien quelque évolution de l'algorithmes A*.

On se base sur cette étude on peut faire le choix de méthode de simulation des individus et les algorithmes de parcoure de chemin pour atteindre notre dut.

Dans le chapitre suivant on va voir la conception de notre projet.

Chapitre 03:

Conception

1. Introduction

Après l'étude détaillée de l'animation et de la simulation comportementale dans les sections précédentes qui va nous aider à choisir le modèle de représentation de l'environnement, d'agent virtuel et les méthodes la simulation, nous allons présenter dans ce chapitre notre solution concernent le problème de déplacement d'un piéton dans un bâtiment en évitent les obstacles et les collisions, on a commencée par exposée et préciser notre objectif.

Ensuite, nous allons donner l'explication de nos choix en ce qui concerne la représentation de l'environnement ainsi que le modèle d'agent virtuel et la planification de chemin en utilisent les algorithmes de recherches, pour atteindre notre but on doit passer par la phase de conception.

2. Objectif

L'objectif principale de notre travail est de concevoir un modèle de simulation du mouvement des piétons dans un bâtiment, et géré le déplacement des piétons autonome dans un environnement Virtual composé de plusieurs niveau vertical en évitent des collisions avec des obstacles statiques ou dynamiques ainsi que La simulation des mouvements soit réaliste.

Dans ce contexte, nous nous intéressons plus particulièrement au comportement de piétons qui se déplacent dans l'étage de même niveau et entre les étages de bâtiment qui représente les niveaux verticaux de bâtiment, pendant le déplacement il rencontre des obstacles statiques et d'autre piéton dont il faut l'éviter. L'évitement de collision et des obstacles consiste à utiliser des techniques de représentation de l'environnement et la planification de chemin.

Pour atteindre cet objectif, nous proposons un modèle qui décrit la topologie de l'environnement et le représenté par le graphe de visibilité et représenté l'évolution de la dynamique d'un agent virtuel par un modèle microscopique. Cette approche microscopique est particulièrement adapté pour simuler de tels types de comportement, car la simulation dans les quelles chaque entité est gérée

individuellement et chaque individu aura son propre fonctionnement.

La planification de chemin à l'aide de l'algorithme A* exécuté sur le graphe de visibilité, dans l'étape de représentation de l'environnement, il faut définir les sommets des polygones décrivant les obstacles de l'environnement, comme points clefs de la carte de cheminement, l'espace libre est donc représenté par un ensemble de points clefs mutuellement visibles.

La planification de chemin par L'algorithme de recherche **A*** qui recherche de chemin dans un graphe entre un nœud initial et un nœud final tous les deux donnés.

3. Conception

L'étape de conception d'un système est un processus qui permet d'élaborer un plan ou un schéma directeur quant à la manière des divers composants et modules du système et leurs interactions dont doit fonctionner, les méthodes d'élaboration des différentes représentations de l'environnement. Un logiciel efficace est un système ou un processus répondant à un besoin en tenant compte des contraintes, le développement d'un système bien conçu est facile à réaliser, à maintenir, facile à comprendre, repose sur la bonne conception .

Dans cette étape la conception de notre projet est constituée de deux parties une conception globale et une conception détaillée.

3.1 Conception globale

La conception globale de notre système déterminé les fonctions majeures. Nous commençons par un plan général montré dans la figure 3.1 pour déterminer la vision globale de l'espace de navigation. La structure globale est définit comme suit : représenter l'environnement via le graphe de visibilité pour générer l'espace navigable et non navigable, deuxième étape est la planification du chemin en utilisent l'algorithme A*, ensuite réaliser le déplacement des agents virtuel selon les comportements individuels et les comportements de groupe.

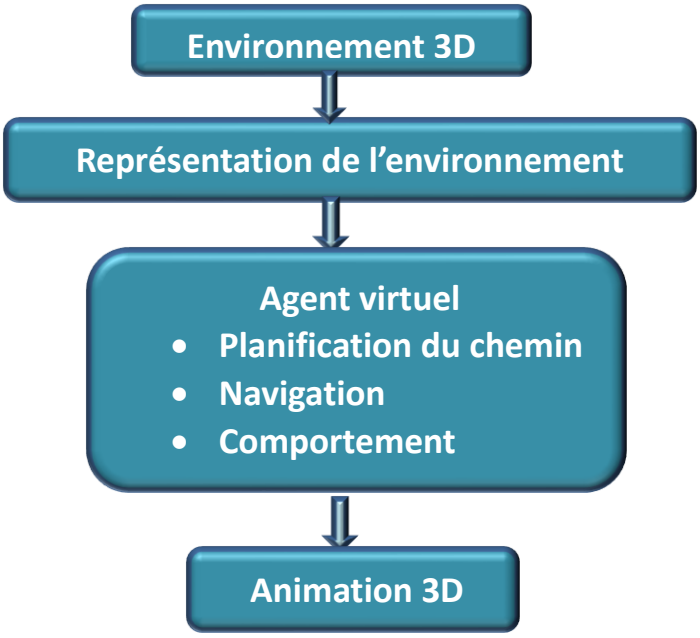


Figure 3.1. La conception globale du système

3.2 Conception détaillée

Dans cette étape, nous allons faire une conception détaillée de notre système, et expliquer en détail la structure et les méthodes utiliser.

Dans chaque niveau du bâtiment on va utiliser un module de représentation de l'environnement 3D, un autre module de planification de chemin et un module pour la représentation des comportements de l'agent virtuel. Entre les niveaux La connexion doit être planifiée le chemin de déplacement vertical.

Ce que on va expliquer dans la section suivante :

3.2.1 Module de représentation d'environnement 3D

Le rôle de ce module consiste de décomposition de l'environnement 3D en utilisent le graphe de visibilité.

Étant donné une scène composée de polygones simples disjoints, le graphe de visibilité de cette scène est le graphe (V, E) tel que :

– V, ensemble des sommets du graphe, est l'ensemble des sommets des polygones ;

– E, ensemble des arêtes du graphe, est l'ensemble des paires e = (p, q) de sommets de V tels que le segment [p, q] a une intersection nulle avec l'intérieur des polygones.

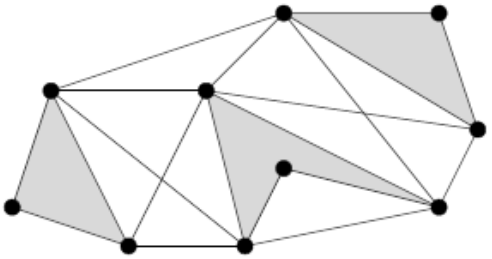


Figure. 3.2. Graphe de visibilité d'une scène. [29]

Les arêtes du graphe de visibilité relient donc les sommets de la scène mutuellement visibles. La figure 3.3 Montre la représentation géométrique du graphe de visibilité, ou les segments correspondant aux arêtes du graphe ont été tracés dans la scène. Le graphe de visibilité contient au moins les arêtes correspondant aux cotés des polygones et aux segments de l'enveloppe convexe de la scène.

Ce graphe n'est pas orienté, mais les arêtes incidentes `a un sommet sont classées dans l'ordre dans lequel sont répartis les segments correspondants dans la scène. Le nombre de sommets de ce graphe est le nombre total n de sommets des polygones [29].

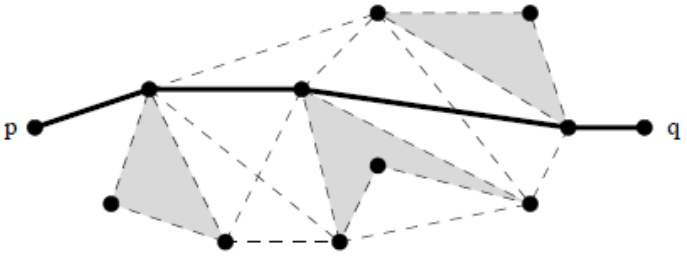


Figure 3.3 Plus court chemin entre deux points en présence d'obstacles.

- Algorithme simplifié de Graphe de visibilité

```
Début
    Installez tous les sommets d'obstacles dans GV, ainsi que la position de
    départ et de but
        Pour chaque paire de nœuds  u, v dans GV faire
            Si le segment (u,v) est sur le bord de l'obstacle alors
                insérer (u,v) dans GV
            Sinon
                Pour chaque bord d'obstacle e  faire
                    Si le segment (u,v) ne coupe pas e alors
                        insérer (u,v) dans GV
                    Fin si
                Fin pour
            Fin si
        Fin pour
    Rechercher VG en utilisant A*
Fin
```

Les graphes de visibilité servent par exemple en planification de trajectoire : dans une scène de polygones, le plus court chemin évitant les obstacles entre deux points p et q emprunte les arêtes du graphe de visibilité (figure 3.3). Le calcul de ce chemin se ramène alors à un calcul de plus court chemin dans le graphe de visibilité, ou le poids des arêtes du graphe est la longueur de l'arête géométrique [30]

3.2.2 Module de planification de chemin

Le déplacement de l'agent virtuels dans l'environnement soit dans le même niveau ou bien entre les déférents les niveaux de la scène, ce déplacement doit être planifié

pour aboutir un chemin on évite les collisions et les obstacles

Chaque niveau de la scène est représenté par un nœud caractérisé par le centre, la longueur et la hauteur, la relation entre les nœuds est représentée par des arcs dont les points de début et fin son des points d'accessibilité entre les déférents niveaux de la scène.

Le déplacement dans le même niveau se fait par un graphe de visibilité discrétisé la scène et permettre de détecté les obstacles statiques, et pour trouver un chemin optimal l'application de l'algorithme de A* est efficace.

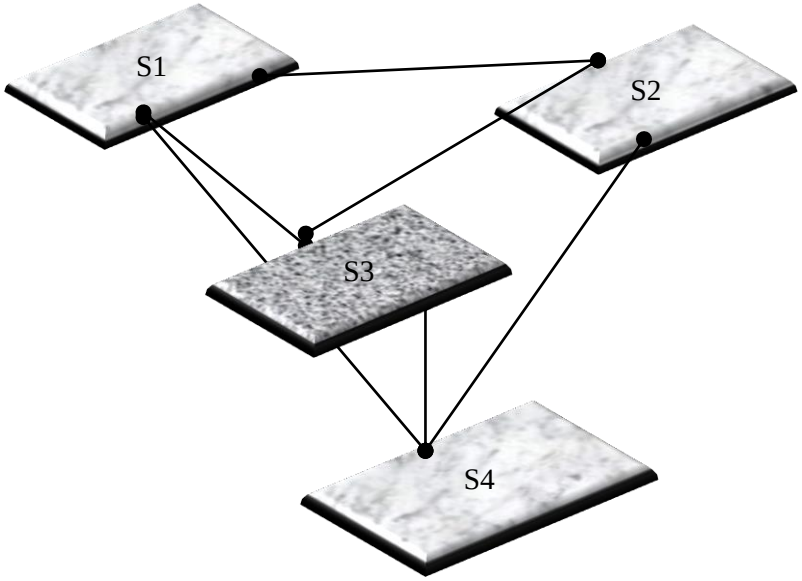


Figure. 3.4. Planification de chemin entre les niveaux du bâtiment

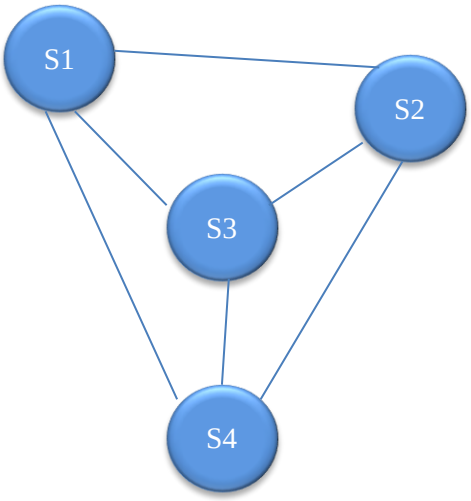


Figure. 3.5. Graphe de Planification de chemin entre les niveaux du bâtiment

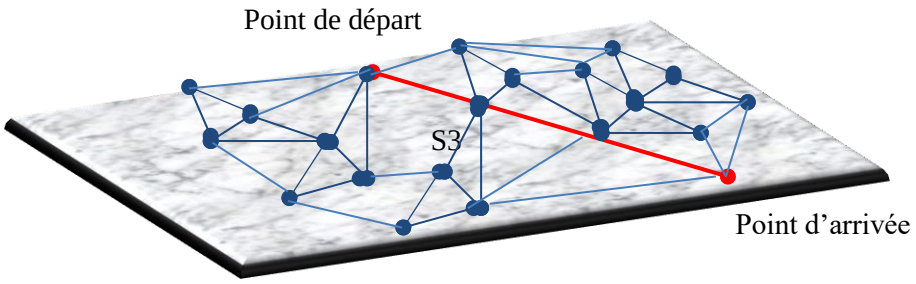


Figure. 3.6. Graphe de visibilité

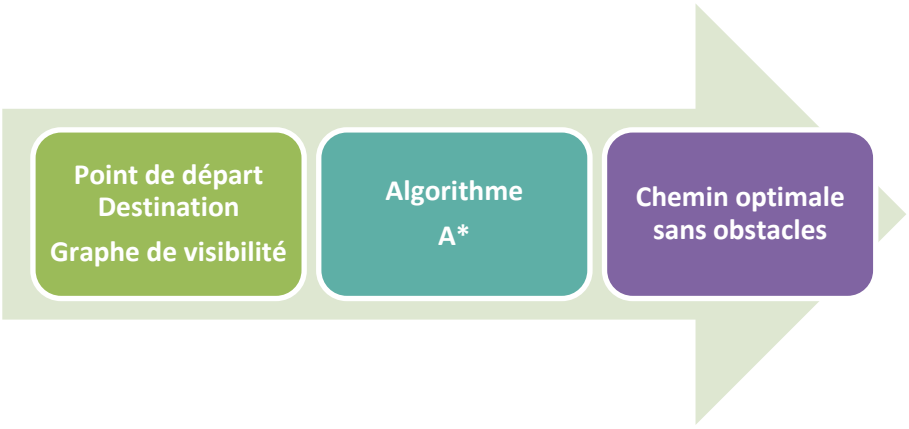


Figure 3.7. La démarche de Planification de chemin dans le même niveau

- **L'algorithme A***

A* commence à un nœud choisi le nœud de départ. Il applique à ce nœud un « coût » (habituellement zéro pour le nœud initial). A* estime ensuite la distance qui sépare ce nœud du but à atteindre. La somme du coût et de l'évaluation représente le coût heuristique assigné au chemin menant à ce nœud. Le nœud est alors ajouté à une file d'attente prioritaire, couramment appelée open list.

L'algorithme retire le premier nœud de la liste prioritaire (en raison du fonctionnement d'une liste, le nœud à l'heuristique la plus basse est retiré en premier). Si la liste est vide, il n'y a aucun chemin du nœud initial au nœud d'arrivée, ce qui interrompt l'algorithme. Si le nœud retenu est le nœud d'arrivée, A* reconstruit le chemin complet et s'arrête. Pour cette reconstruction on se sert d'une partie des informations sauvées dans la liste communément appelé closed list décrite plus bas.

Si le nœud n'est pas le nœud d'arrivée, de nouveaux nœuds sont créés pour tous les nœuds contigus admissibles ; la manière exacte de faire dépend du problème à traiter. Pour chaque nœud successif, A* calcule son coût et le stocke avec le nœud. Ce coût est calculé à partir de la somme du coût de son ancêtre et du coût de l'opération pour atteindre ce nouveau nœud.

L'algorithme maintient également la liste de nœuds qui ont été vérifiés, couramment appelée closed list. Si un nœud nouvellement produit est déjà dans cette liste avec un coût égal ou inférieur, aucune opération n'est faite sur ce nœud ni sur son homologue se trouvant dans la liste. Si un nœud nouvellement produit est déjà marquer comme un nœud vérifier avec un coût égal ou inférieur, aucune opération n'est faite sur ce nœud Après, l'évaluation de la distance du nouveau nœud au nœud d'arrivée est ajoutée au coût pour former l'heuristique du nœud. Ce nœud est alors ajouté à la liste d'attente prioritaire, à moins qu'un nœud identique dans cette liste ne possède déjà une heuristique inférieure ou égale.

Une fois les trois étapes ci-dessus réalisées pour chaque nouveau nœud contigu, le nœud original pris de la liste prioritaire marque l'état nœud vérifié. Le prochain nœud est alors retiré de la liste prioritaire et le processus recommence.

- **Organigramme de l'algorithme A***

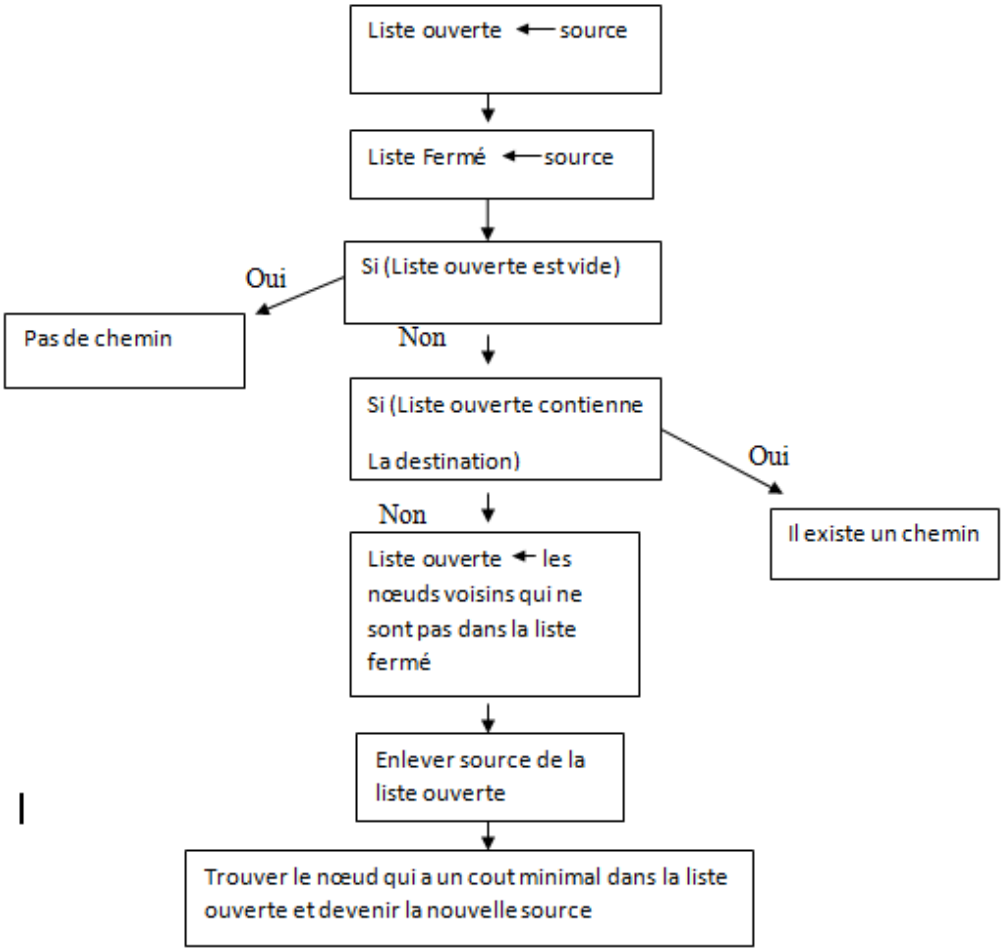


Figure3.5 : Organigramme explicatif pour expliquer A*

Module de l'agent virtuel

L'agent virtuel se déplace d'un point de départ vers une destination précise. Le but est de simuler les mouvements des piétons dans cet environnement virtuel. Ce but est basé sur l'animation comportementale qui est représenté en trois phases suivantes :

- Perception des obstacles et autres individus dans l'environnement, ce qui fournit des informations sur la nature et la position des obstacles.
- En utilise ces informations par le modèle comportemental pour décider de l'action à exécuter, ce qui va produire des paramètres pour une procédure de mouvement.
- Réalisation du mouvement par l'individu, en action finale.

Ces comportements précis se basent aussi sur les caractéristiques de l'agent virtuel suivant : la vitesse, la densité, le flux et la position.

Notre Agent virtuel est représenté selon l'approche microscopique, ou chaque individu a son propre comportement, ses propres décisions et interagit avec les autres, et pendant la navigation il suit un chemin planifier représenté dans l'environnement.

Les comportements traités dans ce projet son des comportements individuels proposés par **Craig Reynolds** qu'on a vu dans les chapitres précédents, ils vont aider l'agent autonome à se déplacer d'une manière cohérente. On va présenter chaque une de ces comportements avec son pseudo algorithme dans la section suivante :

- **Le suivi de chemin**

Le principe sera de déplacer le robot suivant un parcours constitué de deux points ou plus, en l'animant et en le conservant orienté dans la direction dans laquelle il marche.

Il va aussi nous falloir quelques attributs de classe pour stocker certaines informations lors de l'exécution de l'application : la destination en cours du robot ainsi que la direction dans laquelle il doit marcher pour l'atteindre, la vitesse de déplacement et la distance restante avant d'atteindre le point de passage. À partir de ces points de passage rangés dans une file, il ne nous reste qu'à récupérer la direction dans laquelle doit avancer le robot à un moment donné et gérer l'alignement du robot avec la direction dans laquelle il marche. Cette opération sera répéter à chaque fois que l'on cherche une nouvelle destination. Si le déplacement déjà en cours il y a deux choses à faire. La première est de déplacer le robot dans la direction voulue, stockée dans mDirection, et de diminuer la distance restante avant l'arrivée de la distance que l'on vient tout juste de parcourir.
enfin, il faut vérifier si le point de passage est atteint.

- **Algorithme Le suivi de chemin**

Début
  Si un point de passage sera atteint par l'objet **faire**
    Récupéré le prochain point de passage et la direction
  Sinon
    Si déplacement déjà en cours **faire**
      Déplacer le robot dans le direction voulue stocker dans mdirection
        Diminué la distance reste avant l'arrivé à la destination
    Finsi
  Finsi
  Vérifier si le point de passage est atteint
Fin

- **Evitement de collision**

L'évitement de collisions implique la prévision de collisions potentielles et le changement de direction et de vitesse pour les prévenir, elle générer une force de direction permettant d'éviter les agents qui sont sur le même chemin, et même si l'environnement est peuplé d'un grand nombre d'obstacle, ce comportement utilisera un à la fois pour calculer la force d'évitement

- **Pseudo code**

```
Début
Ahead = position +velocité.normalise () * max _see_ahead ;
Ahead_2=pos+vélocité. Normalise ()*max_see_ahead*0.5 ;
Avoid_force= Ahead – le centre de l'obstacle
Avoidance_force= Avoidance_force.normalise()*max avoid force.
Return (force_avoidance)
Fin
```

4. Conclusion

Dans ce chapitre nous avons essayé à détailler la conception de notre projet à travers les deux étapes de conception, qui sont la conception générale et la conception détaillée et nous avons vu que cette étape très importante dans la compréhension de notre système.

Dans la prochaine étape nous allons implémenter ce système pour obtenir un résultat. Ainsi nous allons citer les différents outils qu'on a utilisés pour cette réalisation.

Chapitre 04 :

Implémentation

1. Introduction

Dans ce chapitre on va présenter la phase finale du notre système, celle qui est la phase de l'implémentation. Nous allons présenter l'environnement de programmation, et ensuite nous allons exposer quelques scénarios de mouvement de piéton dans un environnement peuplé par des obstacles et par d'autre piéton pour illustrer notre résultat de simulation.

2. Outils de développements

Pour l'implémentation on a utilisée Visual Studio 10, utilisant la plateforme OGRE 3D pour la modélisation de l'environnement, et l'agent virtuel.

2.1 Ogre 3D (Object-Oriented Graphics Rendering Engine)

OGRE est un moteur 3D open-source multiplateforme (Linux, Win32, MacOSX, IPhone) orienté scène qui permet à partir d'objets à facettes de réaliser un environnement tridimensionnel qui sera perçu par un rendu bidimensionnel au travers d'une ou plusieurs caméras virtuelles.

Ogre est un moteur de rendu 3D en temps réel, orienté sur les scènes, le logiciel est open source .La première version de logiciel à été publié en février 2005, le logiciel est écrit en c++ et à été porté sur Windows, Linux et Pocket PC et autre Systèmes d'exploitations.

2.2 Les Caractéristiques :

- Prise en charge de Direct3D 9 & 11, Metal, OpenGL (y compris ES2, ES3 et OGL3+) et WebGL (Emscripten)
- Prise en charge de Windows (toutes les versions principales et WinRT), Linux, Mac OSX, Android et iOS
- Interface OO simple et facile à utiliser conçue pour minimiser l'effort requis pour rendre les scènes 3D et pour être indépendante de l'implémentation 3D, c'est-à-dire Direct3D/OpenGL.

- Conception propre et épurée et documentation complète de toutes les classes de moteurs
- Un puissant langage de déclaration de matériel vous permet de conserver des actifs matériels en dehors de votre code
- Les textures peuvent être fournies et mises à jour en temps réel par des plugins, par exemple un flux vidéo
- Prise en charge sophistiquée de l'animation squelettique
- Animation de SceneNodes pour les chemins de caméra et techniques similaires, en utilisant l'interpolation spline si nécessaire

3. Architecture de l'application

3.1 Création de l'environnement

- La mise en place de la scène:

Premièrement on a créé trois plans représente des différents niveaux verticales

➢ La création de plans

```
Plane plan(Vector3::UNIT_Y,0);
MeshManager::getSingleton().createPlane("sol",
ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME, plan, 2000, 2000, 1, 1,
true, 1, 1, 1, Vector3::UNIT_Z);
```

L'entité doit être créée par le SceneManager pour pouvoir être ajoutée à la scène. La méthode createEntity() permet justement de faire tout cela

```
Entity *ente= mSceneMgr->createEntity("EntiteSol", "sol");
```

➢ Créer le nœud de la scène

Un SceneNode est un objet invisible auquel on va pouvoir attacher un nombre indéfini d'entités On devra passer par un noeud déjà existant, ce qui va nous permettre d'avoir des relations d'héritage entre nos noeuds. Et on va attacher les entités à nos noeuds, ceux-ci peuvent avoir un noeud parent, lui-même enfant d'un autre noeud, et ainsi de suite jusqu'au noeud appelé « racine » de la scène.

la méthode getRootSceneNode() nous permet de récupérer un pointeur sur le noeud racine unique de la scène. On appelle ensuite sa méthode createChildSceneNode pour lui ajouter un nouveau noeud fils.

```
SceneNode *pos = mSceneMgr->getRootSceneNode()->createChildSceneNode();
        pos->attachObject(ente);
        ente->setMaterialName("Examples/Floor");
```

- Création des obstacles

Nous avons placé des objets sur la scène pour peupler l'environnement avec des obstacles. Cela nous permettra de voir l'agent virtuel se déplacer par rapport aux autres objets à l'écran en évitent ces objets.

```
ent = mSceneMgr->createEntity("Knot11", "WoodenChair.mesh");
node = mSceneMgr->getRootSceneNode() -
>createChildSceneNode("KnotNode11",Vector3(-600, 10, 500));
 node->attachObject(ent);
  node->setScale(0.2,0.2,0.2);
```

un objet créer dans le deuxième plan avec une taille différente

```
ent2 = mSceneMgr->createEntity("Knot22", "WoodenChair.mesh");
    node2 = mSceneMgr->getRootSceneNode()-
>createChildSceneNode("KnotNode22",Vector3(1000, 450, -500));
    node2->attachObject(ent2);
    node2->setScale(0.1,0.1,0.1);
```

Résultat obtenu :

Figure 4.1 une scène de trois plans vue du haut

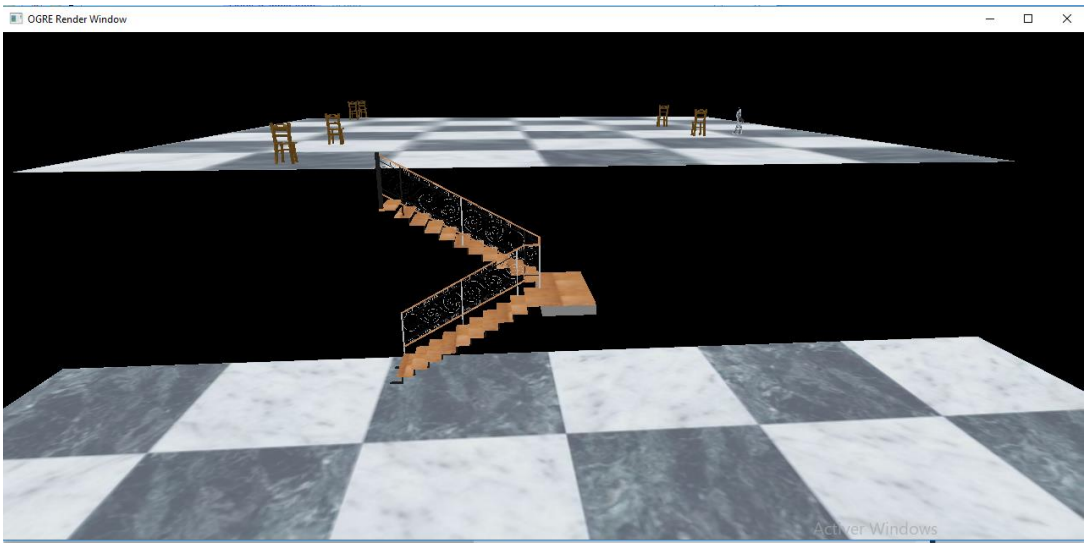


Figure 4.2 une scène de deux niveaux vue de coté

Enfin, nous voulons régler la caméra sur un bon point de vue pour voir cela. Nous allons déplacer la caméra pour obtenir une meilleure position.

- Réglez la caméra pour regarder notre travail

```
mCamera-> setPosition (90.0f, 280.0f, 535.0f);
mCamera-> pas (Degré (-30.0f));
mCamera-> lacet (Degré(-15.0f));
```

3.2 Représentation de l'environnement par le graphe de visibilité

La discrétisation de la scène on utilise le graphe de visibilité par la création de la class CGraphe qui est constitué des attributs et des méthodes suivantes :

```
class  CGraphV {

          // Méthodes
          public:

     CGraphV(Ogre::SceneManager * MSG)// constructeure
        {
                      mSceneMgr=MSG;
        }

       int WhichSide(CNode* v1, CNode* v2,CNode* v)

       bool Intersect(CNode* p1,CNode* p2)

       void Build_Edges()

       void GraphDraw(Ogre::ColourValue C,Ogre::String S,int HT)

         ///// attribut//////

       std::vector<CNode*> Vertices;
       std::vector<CEdge*> Edges;
       int NB_obs;
       int NB_edgO;
       Ogre::SceneManager *mSceneMgr;
```

Expliquons en détaille les méthodes de la classe CGraphe et son utilité.

Commençons par la méthode `int WhichSide(CNode* v1, CNode* v2,CNode* v)`

```
int WhichSide(CNode* v1, CNode* v2,CNode* v)
 {
double x1,x2,xp;
double y1,y2,yp;
double Eq;

             xp=v->getx();
             yp=v->gety();

             x1=v1->getx();
             y1=v1->gety();

             x2=v2->getx();
             y2=v2->gety();

        Eq = ((yp - y1) * (x2 - x1)) - ((y2 - y1) * (xp - x1));

        if ( Eq > 0) return(-1);
             if ( Eq == 0) return(0);
             if ( Eq < 0) return(1);
   }
```

détermine de quel côté d'une ligne se trouve le point (xv,yv). La ligne va de (x1,y1) à (x2,y2) la méthode renvoie (-1) pour un point à gauche ,(0) pour un point sur la ligne et (+1) pour un point à droite.

Pour tester l'intersection entre les arcs on a utilisé la fonction booléen

```
bool Intersect(CNode* p1,CNode* p2)
      {
            for (int i=0;i<(NB_obs*4);i++)

            {
                  int a,b,c,d ;
                  a=WhichSide(p1,p2,Edges[i]->getV1());
                  b=WhichSide(p1,p2,Edges[i]->getV2());
                  c=WhichSide(Edges[i]->getV1(),Edges[i]->getV2(),p1);
                  d=WhichSide(Edges[i]->getV1(),Edges[i]->getV2(),p2);

                  if (( a*b==-1 )&&(c*d==-1))
                  {
                        return true;
                  }
                                    }
            return false;
      }
```

Elle renvoi « true » si 'il y a une intersection et si non « false ».

Ensuite on a appliqué le teste d'intersection entre les arcs formé par la liés on des points de la scène qui ne sont pas des bordures des obstacles, et on construit les arcs ou les Edjes en utilisent la méthode `Build_Edges()`.Donc on a construire un table « Edges » contient les arcs non intersecté.

```
void Build_Edges()
       {
             for (int i=0;i<Vertices.size();i++)
             {
              for (int j=i+1;j<Vertices.size();j++)
                    {
                 if ( Vertices[i]->getNObs()!=Vertices[j]->getNObs())
                           {
                           if (Intersect(Vertices[i],Vertices[j])== false)

                                 {
                                Edges.push_back(new
CEdge(Vertices[i],Vertices[j],0));
                                 }
                           }
                    }
              }
```

L'étape suivante est de crée un objet G1 de type CGrapheV* pour la construction des arcs (Edjes). La requêtes de scène get Bounding-Box utilisent des boîtes englobantes alignées sur les axes: nous fournissons deux vecteurs 3D qui définissent les coins opposés dans l'espace 3D, et Ogre fournit tous les objets qui se trouvent à l'intérieur de cette boîte. Nous avons ajouté un espace de sécurité a la boite englobante avec l'instruction suivante `v=v+ Ogre::Vector3(20,20,20);`

```
CGraphV* G1;

         G1=new CGraphV(mSceneMgr);//création de l'objet
         G1->NB_obs=Ent_Obs.size();

                Ogre::Vector3 v,v1;

         G1->Vertices.push_back(new CNode(-820,600,0,NULL,-1));//point initiale
         G1->Vertices.push_back(new CNode(980,-360,0,NULL,0));//point finale

         int j=2;//0,1 ocupé par le point initiale et le point finale
         for (int i=0;i<Ent_Obs.size();i++)
         {
                v=Ent_Obs[i] ->getBoundingBox().getHalfSize();
               v=v*0.2;
               v=v+ Ogre::Vector3(20,20,20);

              v1=Node_Obs[i]->getPosition();

              G1->Vertices.push_back(new CNode(v1.x-v.x, v1.z-v.z, 0, NULL,
i+1));//les coins de chaque obstacle
              G1->Vertices.push_back(new CNode(v1.x+v.x, v1.z-v.z, 0, NULL, i+1));
              G1->Vertices.push_back(new CNode(v1.x+v.x, v1.z+v.z, 0, NULL, i+1));
              G1->Vertices.push_back(new CNode(v1.x-v.x, v1.z+v.z, 0, NULL, i+1));


              G1->Edges.push_back(new CEdge(G1->Vertices[j], G1->Vertices[j+1],
i+1));     //les edges de chaque obstacle
              G1->Edges.push_back(new CEdge(G1->Vertices[j+1] ,G1->Vertices[j+2],
i+1));
              G1->Edges.push_back(new CEdge(G1->Vertices[j+2],G1-
>Vertices[j+3],i+1));
              G1->Edges.push_back(new CEdge(G1->Vertices[j+3],G1-
>Vertices[j],i+1));
              j=j+4;
         }

         G1->Build_Edges();
```

ensuite dessiner le Graphe de visibilité avec la méthode GraphDraw`(Ogre::ColourValue C,Ogre::String S,int HT)`

```
void GraphDraw(Ogre::ColourValue C,Ogre::String S,int HT)
      {

       std::vector <CEdge*>::iterator it;
       Ogre::SceneNode *PathNode;

       if (Edges.empty()==false)
```

```
       {
            Ogre::ManualObject*  Chemin = mSceneMgr-
>createManualObject(S);
            Chemin->begin("BaseWhiteNoLighting",
Ogre::RenderOperation::OT_LINE_LIST );
            Chemin->colour(C);
            for (it=Edges.begin(); it!=Edges.end();it++)
            {

                  Chemin->position((*it)->getV1()->getx(),HT,(*it)-
>getV1()->gety());
                  Chemin->position((*it)->getV2()->getx(),HT,(*it)-
>getV2()->gety());


            }

            Chemin->end();
            Chemin->convertToMesh(S);
            PathNode =mSceneMgr->getRootSceneNode()-
>createChildSceneNode(S);
            PathNode->attachObject(Chemin);

       }
      }
```

Résultat obtenus:

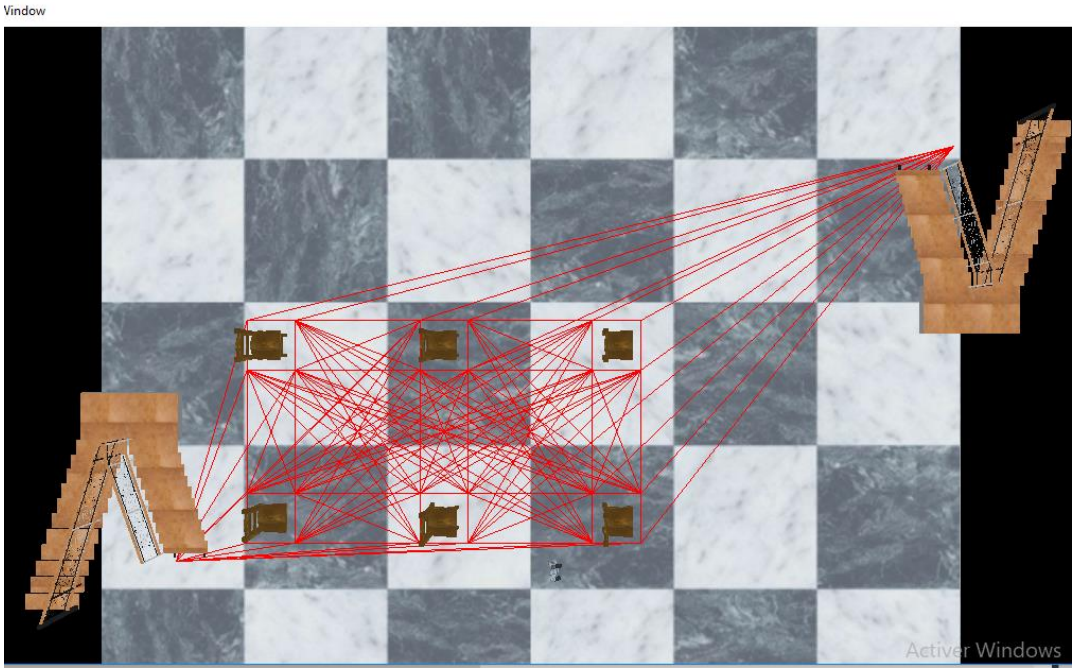


Figure 4.3 le graphe de visibilité

3. Création de l'agent virtuel

➢ La méthode de création de l'agent virtuel :

La création d'un robot représente notre agent virtuel ce fait par la méthode

void createRobot(String name, Real x, Real y, Real x1,Real y1)

nous avons créé l'entité pour le robot, puis créer un SceneNode auquel il pourra pendre.

```
mEntity[nbr_robot] = mSceneMgr->createEntity(name, "robot.mesh");
mNode[nbr_robot] = mSceneMgr->getRootSceneNode()->createChildSceneNode("Node"+name);
ensuite on a fait la définition la position du nœud  et l'attachement de l'entité et le nœud
mNode[nbr_robot]-> setPosition(Vector3(x, 10.0f, y));
mNode[nbr_robot]->attachObject(mEntity[nbr_robot]);
```

Résultat :

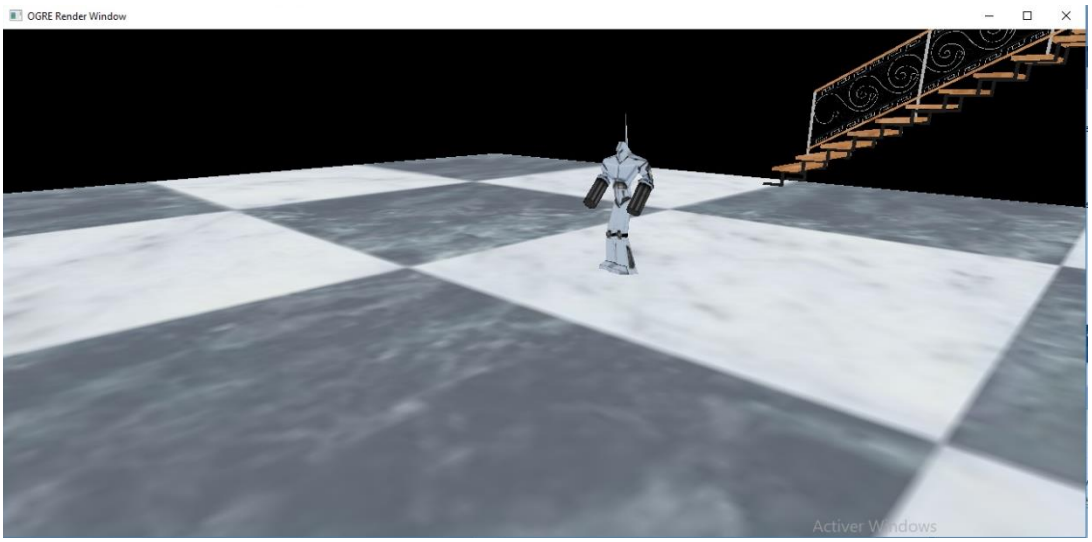


Figure 4.4 l'agent virtuel

➢ La planification de chemin par A*

- La méthode de création de l'algorithme A* :

Pour la planification de chemin on utilise la méthode /// la classe d'abord

`void Astar::Calculer()`

```
void Astar::Calculer()//Vertices* v1,Vertices *v2)
{

	CNode* p=NULL;
	CNode* V1=NULL;
	CNode* V2=NULL;
	CNode* V=NULL;



	liste_O.push_back(m_source);
	if(Egale(m_source, m_dest))
	{
		printf("le noeud de depart est le noeud d'arrivée");
			m_resultat.push_back(m_dest);



	}
	else
	{
	while(liste_O.empty()==false && m_trouve==false   )
	{
//		int i;


		p=liste_O.back();//lire dernier elément dans p
		liste_O.pop_back();//supprimer de la liste
		p->m_etat=1; //



		for(int i =0; i<G->Edges.size(); i++)// pour chaque voisin
		{

			V1=G->Edges[i]->getV1();
			V2=G->Edges[i]->getV2();
			if (p==V1) V=V2;
			if (p==V2) V=V1;

		   if(p==V1||p==V2)
		   {

			if(V==m_dest)
			{
				m_trouve=true;  //noeud destination existe dans la
liste ouverte
				break;
			}



			if (V->m_etat != 1)//  voisin et n'est pas encore visiter
```

```
                    {

                            double dx,dy,d;
                            double CG,CH,CF;

                            dx=V->getx()-p->getx();
                            dy=V->gety()-p->gety();
                            d=dx*dx+dy*dy;

                            CG= p->getcoutG()+d; //CGle cout de distance du point
départ au de point courente
                            CH=calculerH(V);//calculer le cout a partire de point
de départ jusqu'au point d'arrivée
                            CF=CG+CH;


                                    // comparer G
                            int b=0;
                                    std::vector<CNode*>::iterator it;

                                            for (it=liste_O.begin();
it!=liste_O.end();it++)// a revoir
                                           {

                                                  if(V==(*it))
                                                   {
                                                          b=1;
                                                         break;
                                                   }
                                           }
                                    if(b==1)
                                    {

                                            if(CG < (*it)->getcoutG())
                                            {

                                                    // mettre a jour *it
                                                    (*it)->setcoutG(CG);
                                                    (*it)->setcoutF(CF);
                                                    (*it)->setparent(p);

                                            }

                            }
                            else// n'existe pas
                            {
                                    // ajouter pp list O

                                    V->setparent(p);

                                    V->setcoutH(CH);
                                    V->setcoutG(CG);
                                    V->setcoutF(CG+CH);

                                    liste_O.push_back(V);
                                    sort(liste_O.begin(),liste_O.end(),comparaison);
                            }


                    }// end if(T!=-1)
            }//end if numC
```

```
            }//end for


      }//end while

               // suite de la sol
      if( m_trouve==true)
      {
            m_resultat.push_back(m_dest);
             while(p->getparent()!=NULL ) //
             {
                   m_resultat.push_back(p);
                   p=p->getparent();
             }
             m_resultat.push_back(m_source);

             //affichage_Mresultat();
      }
      else
      {
            printf(" la solution n'existe pas");
      }

      }
}
```

Résultat obtenu :

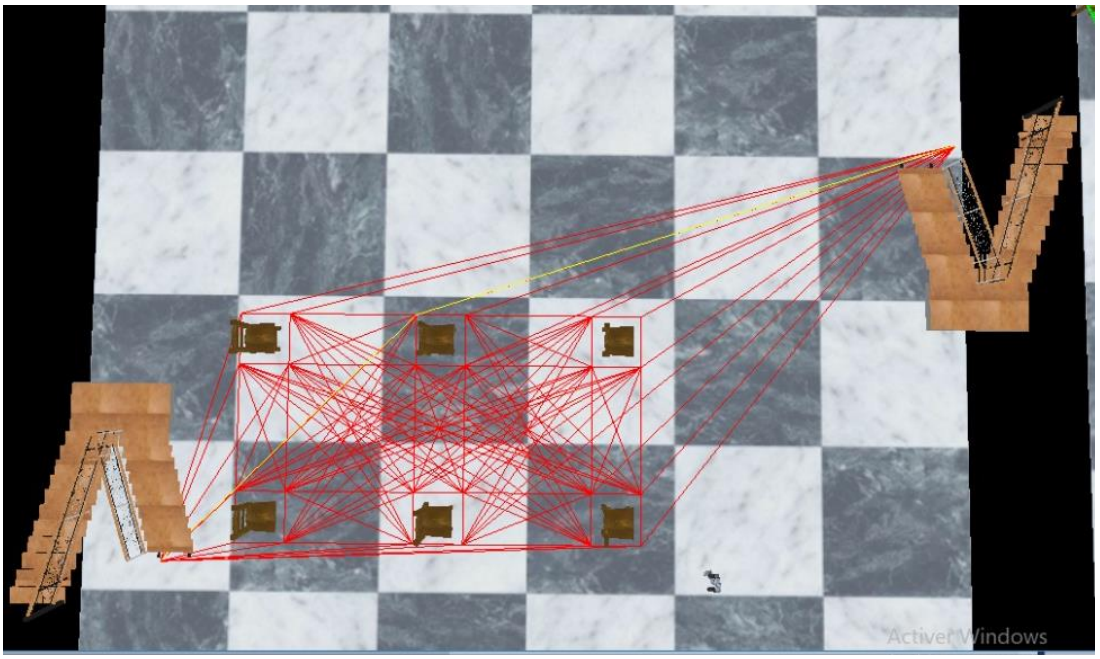


Figure 4.5 l'application de l'algorithme A*sur un plan(ligne en jaune)

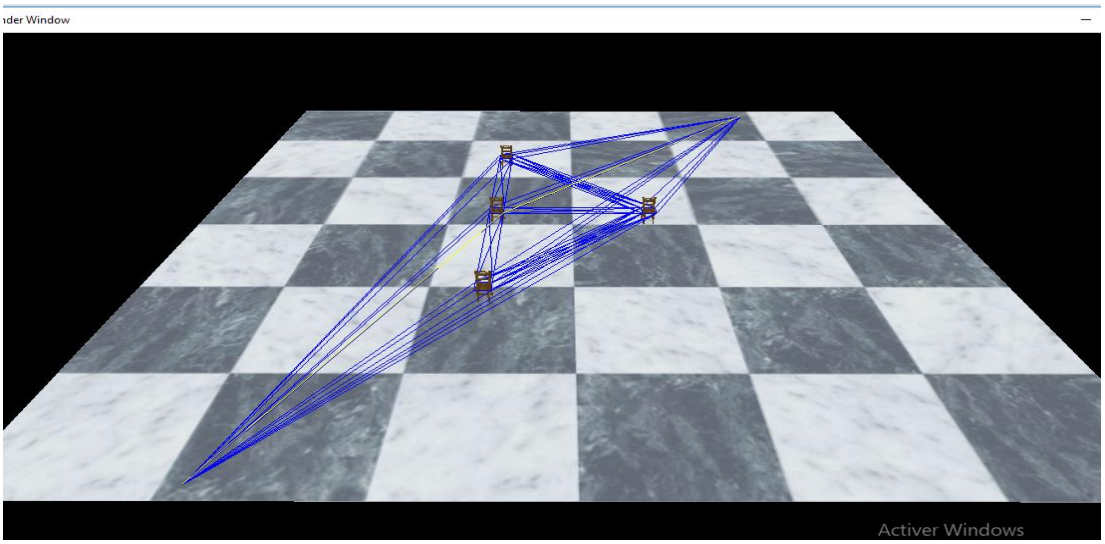


Figure 4.6 l'algorithme A*avec un nombre moins d'obstacles

➢ Navigation

- Suivie de chemin

Cette fonction est appelée pour démarrer le déplacement de l'objet vers la position suivante dans mWalkList.

```
bool nextLocation(int i)
    {

              if ( mWalkList[i].empty() )
             return false;
        mDestination[i]= mWalkList[i].front( );//cela obtient le devant de deque
         mWalkList[i].pop_front( ); // suprimé l'avant de la deque
        mWalkList[i].push_back( mDestination[i]);
         mDirection[i] = mDestination[i] - mNode[i]->getPosition( );
         mDistance[i] = mDirection[i].normalise( );


         return true;

    } //prochain emplacement
```

```
bool frameStarted(const FrameEvent &evt)
    {
              for ( int i=0; i<2;i++)
              {
```

```
              if (mDirection[i] == Vector3::ZERO)
      {
          if (nextLocation(i))
          {
              //définir l' animation marcher " walking animation"
              mAnimationState[i] = mEntity[i]->getAnimationState("Walk");
              mAnimationState[i]->setLoop(true);
              mAnimationState[i]->setEnabled(true);

          }
      }
            else
       {
           Real move = mWalkSpeed[i] * evt.timeSinceLastFrame;
           mDistance[i] -= move;

              if (mDistance[i] <= 0.0f)
           {
               mNode[i]->setPosition(mDestination[i]);
               mDirection[i] = Vector3::ZERO;
// définir l'animation en fonction du fait que le robot un autre point vers lequel
marcher
               if (! nextLocation(i))
               {
                   // définir l'animation  Idle
                   mAnimationState[i] = mEntity[i]->getAnimationState("Idle");
                               mAnimationState[i]->setLoop(true);
                   mAnimationState[i]->setEnabled(true);

                         }
               else
               {
                         }
           }
              else
           {
               mNode[i]->translate(mDirection[i] * move);
           } // else
       } // if
             mAnimationState[i]->addTime(evt.timeSinceLastFrame);//mise a jour du
temp passe de frame pricident

             } //end for

       return ExampleFrameListener::frameStarted(evt);
   }
```

```
// créer la liste de marcher
      mWalkList[0].push_back(Vector3(550.0f,  0.0f,  50.0f ));
      mWalkList[0].push_back(Vector3(-100.0f,  0.0f,  -200.0f ));


        /////
      mWalkList[1].push_back(Vector3(-100.0f,  0.0f,  -200.0f ));
      mWalkList[1].push_back(Vector3(550.0f,  0.0f,  50.0f ));
```

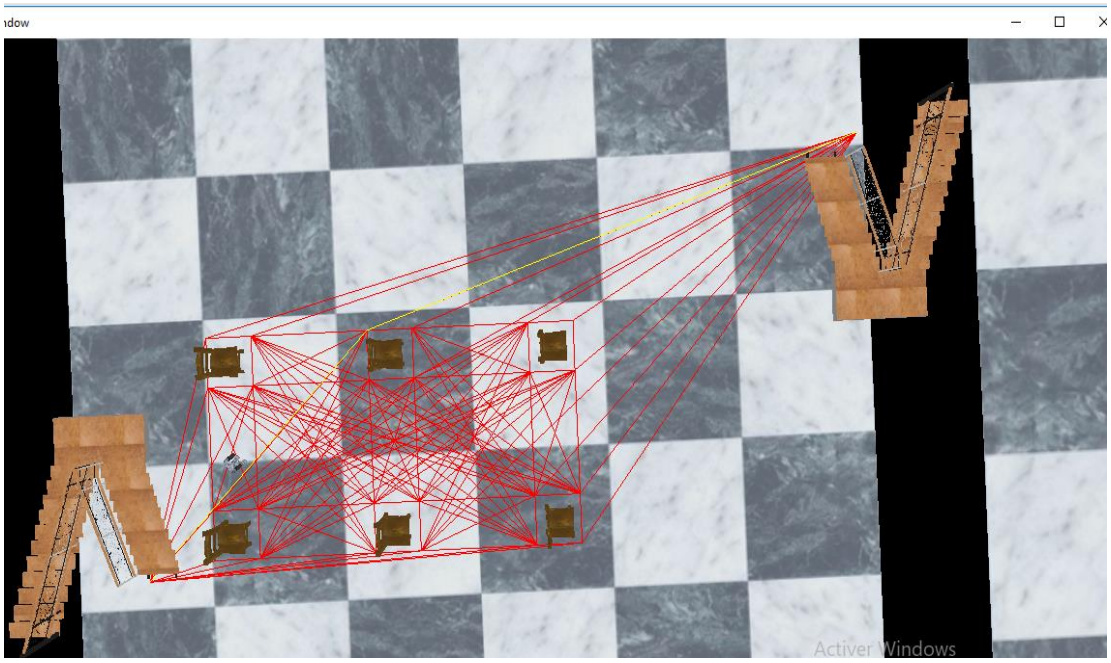


Figure 4.7 la marche du point de départ vers le point d'arrivé

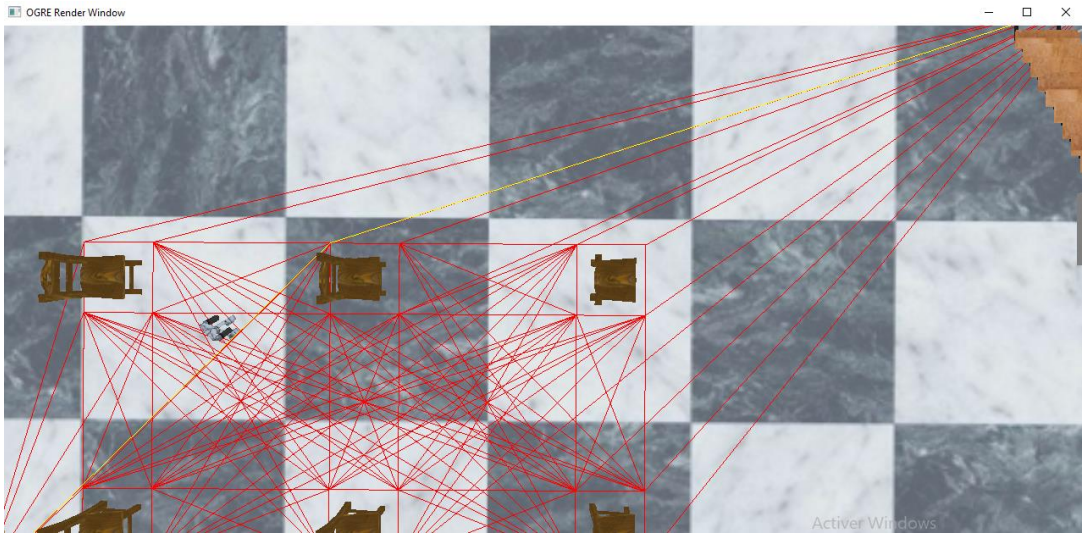


Figure 4.8 le suive le chemin du point de départ vers le point d'arrivé

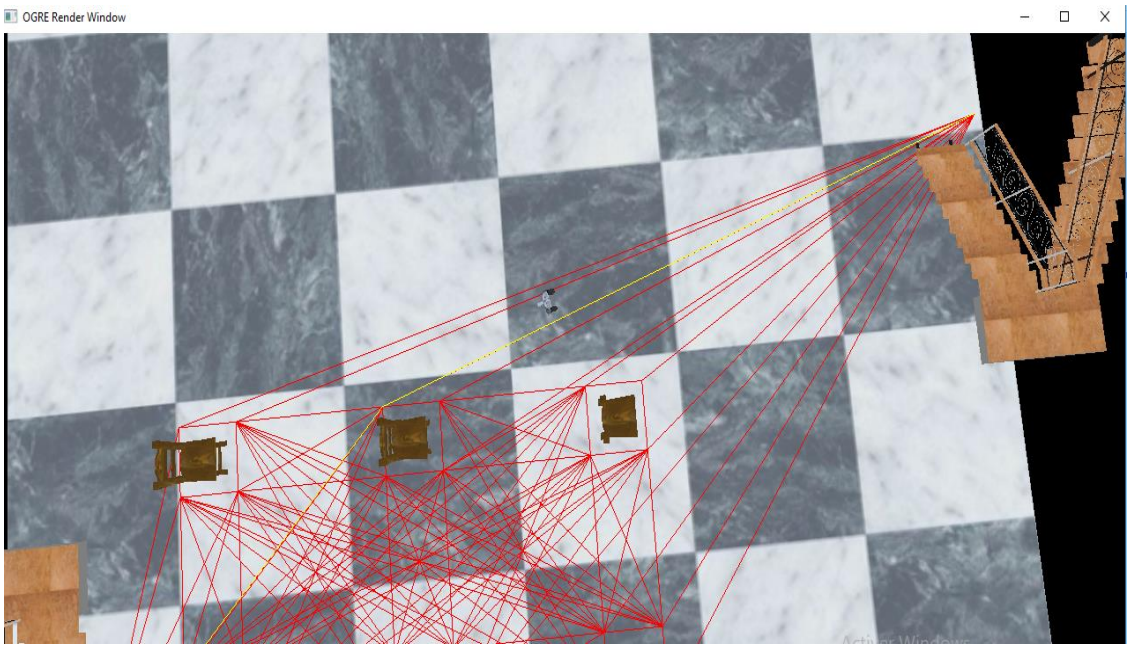


Figure 4.9 le robot se rapproche du point d'arrivé

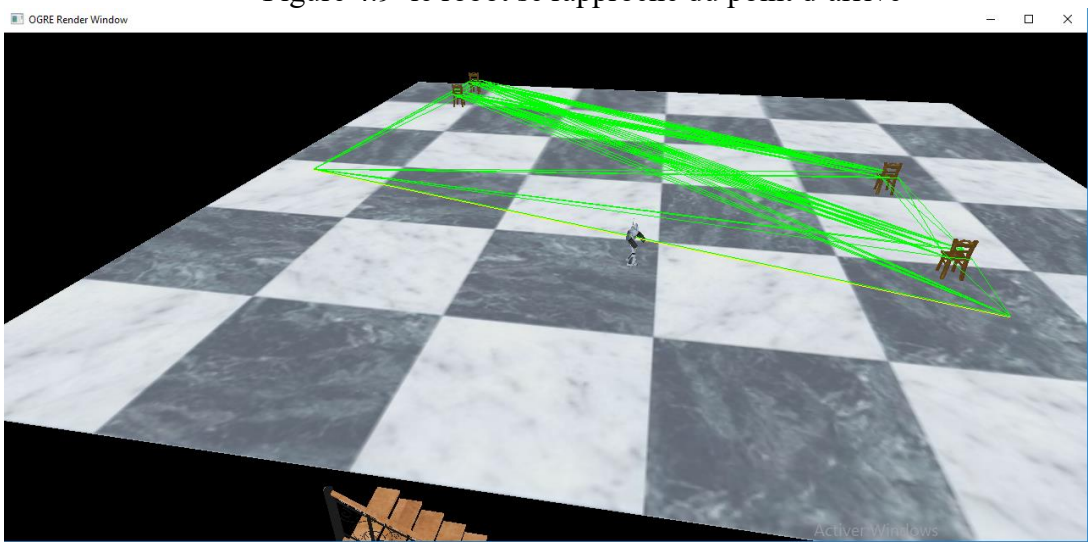


Figure 4.10 le déplacement du robot dans le 2^{iem} étage

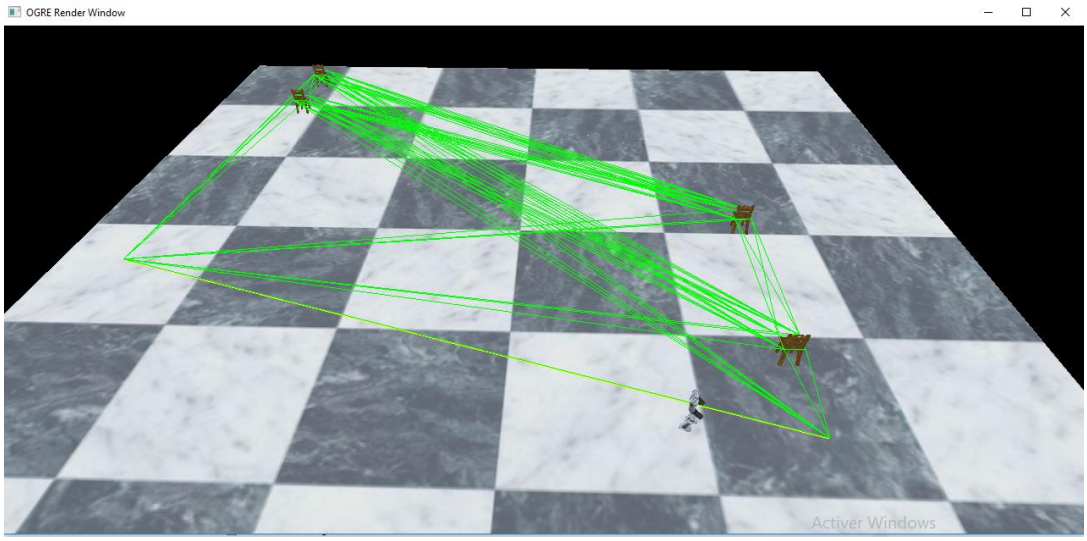


Figure 4.11 du point de départ vers le point d'arrivé dans le 2iem étage

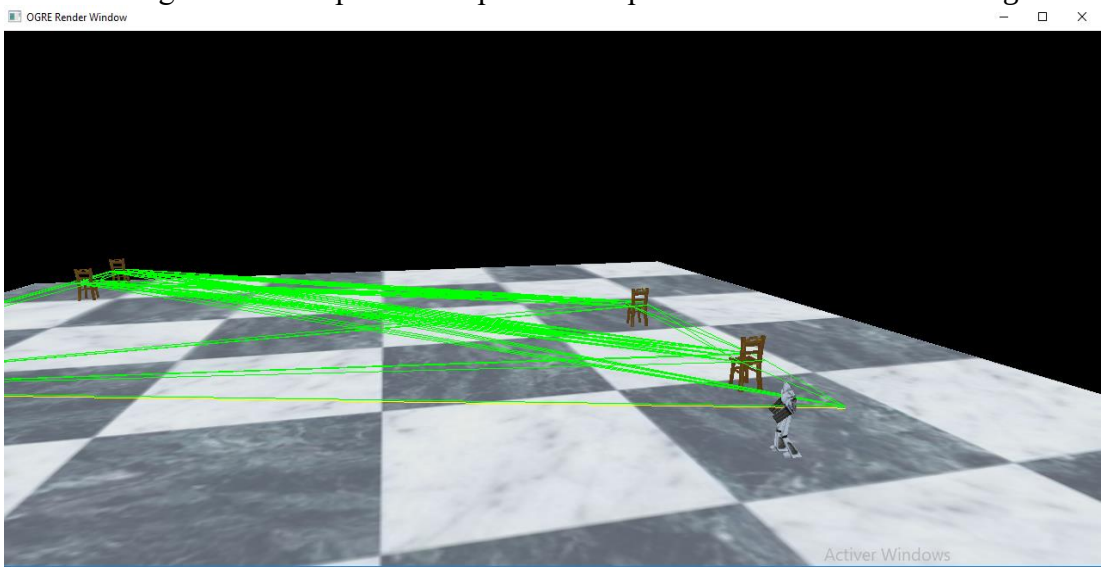


Figure 4.12 l'arrivé du robot et reprenez le chemin et fait le retour

- Evitement de collision

"scene query" tyep Sphere

```
void animation(int i){


// définir l'animation walking
	mAnimationState[i] = mEntity[i]->getAnimationState( "Walk" );
	mAnimationState[i]->setLoop( true );
	mAnimationState[i]->setEnabled( true );


	Vector3 src2 = mNode[i]->getOrientation( ) * Vector3::UNIT_X;
	if ( (1.0f + src2.dotProduct( mDirection[i] )) < 0.0001f )
	{
		mNode[i]->yaw( Degree(180) );
	}
```

```
       else
          Ogre::Quaternion quat = src2.getRotationTo( mDirection[i] );
          mNode[i]->rotate( quat );
       }
```

```
void Animation2(int i){
       bool mRobot=false;
       mNode[i]->setPosition( mDestination[i] );
              mDirection[i] = Vector3::ZERO;
// définir l'animation en fonction du fait que le robot un autre point vers lequel
marcher
                 if (! nextLocation(i) )
                     {
              // définir l'animation Idle
                     mAnimationState[i] = mEntity[i]->getAnimationState( "Idle" );
                     mAnimationState[i]->setLoop( true );
                     mAnimationState[i]->setEnabled( true );

                     }
                     else
                              {
                     Vector3 src = mNode[i]->getOrientation( ) * Vector3::UNIT_X;
                     if ( (1.0f + src.dotProduct( mDirection[i] )) < 0.0001f )
                            {
                            mNode[i]->yaw( Degree(180) );
                            }
                            else
                            {
              Ogre::Quaternion quat = src.getRotationTo( mDirection[i] );
                            mNode[i]->rotate( quat );

                            } // else
                     }
       }

    bool frameStarted(const FrameEvent &evt)
    {
       for (int i=0;i<nbr_robot;i++)
       {
              if (mstate[i]==0)
              {
                     if ( mDirection[i]== Vector3::ZERO )
                     {
                            if(nextLocation(i)){
                                   animation(i);
                            }
                     }
```

```
                    else
                    {
                           Real move = mWalkSpeed[i] * evt.timeSinceLastFrame;
                           mDistance[i] -= move;

                    if (mDistance[i] <= 0.0f)
                           {
                                  Animation2(i);
                           }
                     else
                           {




//-------------------------------------------------------
              // Réglage du "scene query" tyep Sphere
              Vector3 EntPos = mNode[i]->getPosition();
              Vector3 vec1 = mEntity[i]-
>getBoundingBox().getHalfSize();//.length();
              vec1.y=0;
              Real len=vec1.length()+5;
       EntPos=EntPos +(mDirection[i] * len + mDirection[i] *10);// centre du sphere

              Sphere EntSphere(Vector3(EntPos.x, 10, EntPos.z),8);
              mSphereSceneQuery->setSphere(EntSphere);
              // exécute la scene query
              SceneQueryResult &result = mSphereSceneQuery->execute();
              SceneQueryResultMovableList::iterator it = result.movables.begin();

              // obtenir le résultat


              if (it!=result.movables.end())// Evitement
              {
              //mState[i]=2;
              Vector3 vec,pos;
              int alpha;
                     //mWalkList[i].clear();
                     alpha=45;
                     mNode[i]->yaw( Degree(alpha) );
                     vec=mNode[i]->getOrientation( )* Vector3::UNIT_X;
                     vec.normalise();


                     pos=mNode[i]->getPosition()+50*vec;
                     mWalkList[i].push_front(mDestination[i]);
                     mDestination[i] = pos;


                     mDirection[i] = mDestination[i] - mNode[i]->getPosition( );
                     mDistance[i] = mDirection[i].normalise( );
                     mstate[i]=1;

                           }
                           else // Avencement
```

```
                        {
                                mNode[i]->translate( mDirection[i] * move );
                        }
                        mSphereSceneQuery->clearResults();
                                //_____________________________________

                        } // else
                } // if
        }
        else // le robot est en phase d'evitement
        {
                if ( mDirection[i]== Vector3::ZERO )
                {
                        if ( nextLocation(i) )
                        {
                                mstate[i]=0;
                                // définir l'animatio walking
                                animation(i);
                                //---------------------
                        }
                }
                else
                {
                        Real move = mWalkSpeed[i] * evt.timeSinceLastFrame;
                        mDistance[i] -= move;

                if (mDistance[i] <= 0.0f)
                        {
                                mstate[i]=0;
                                Animation2(i);
                        }
                 else
                 {
                        mNode[i]->translate( mDirection[i] * move );
                 }
                }

        }// end else state

        mAnimationState[i]->addTime( evt.timeSinceLastFrame );
   }//end for
    return ExampleFrameListener::frameStarted(evt);
}
```

4. Conclusion

Dans ce chapitre, nous avons présenté la dernière étape de notre modèle où nous avons vu la plateforme OGRE 3D pour la modélisation de l'environnement, et l'agent virtuel, ensuite nous avons détaillé l'architecture de l'application par mentionner les méthodes et les fonctions que nous avons appliqué pour la représentation et la simulation du mouvement de piéton dans l'environnement, et enfin nous avons présenté les résultats obtenu

Conclusion générale

Conclusion générale

Dans ce mémoire nous avons essayée de traité du problème de la simulation du mouvement des piétons dans un bâtiment pour générer l'animation comportementale, et le déplacement autonome des piétons dans un environnement Virtual composé de plusieurs niveau vertical en réalisons une simulation plus proche de la réalité.

Pour assurer le déplacement des éléments nous avons fait la représentation topologique de la structure de l'environnement par définir l'espace occupé par les obstacles ainsi que l'espace libre en utilise la méthode de graphe de visibilité. Cette méthode permettre l'évitement des obstacles, et, pour augmenter la sécurité des déplacements nous avons ajouté une distance entre l'obstacle et l'espace détermine le graphe de visibilité. La planification de chemin nous avons choisir l'algorithme A* pour que la première solution trouvée soit la meilleure, et privilégiant la vitesse de calcul sur l'exactitude des résultats sa donne plus de réalisme au mouvement de l'humain virtuel.

Dans la simulation des comportements nous avons employées la technique d'optimisations SphereSceneQuery du moteur de rendu OGRE 3D pour la détection des collisions.

D'après ces méthodes utilisées, nous avons obtenu comme résultat les comportements de suivi le chemin, évitement des obstacles et des collisions des agents dans l'environnement virtuel.

L'animation comportementale qui est le but principale de notre projet est composée de trois éléments; l'environnement virtuel, l'agent virtuel et la simulation comportementale et bas é sur la boucle d'animation a été partiellement réalisé et cela apparaît dans les méthodes de représentation et la planification de chemin, mais il reste quelque perspectives afin de le complété et de l'améliorer le travail d'une meilleure façon.

Références

[1] S. Donikian, *Modélisation, controle et animation d'agents virtuels autonomes évoluant dans des environnements informés et structurés,* Renne, University of Renne 1, 2004.

[2] R. Bouville, Interopabilité des environnements virtuels 3D,Modèle de reconcilaiation des contenus et des composants logiciels, Insa de rennes, 2012.

[3] l. SIAMES-IRISA, *l'animation comportementale,* Rennes www.game-reserarch.com, www.igda.org/writing/virtuelStorytelling.html.

[4] M. P. Carreno, Simulation comportementale dans les environnements virtuels, 2012.

[5] Mezati Messoud, *Une nouvelle approche pour la sémantique des environnements virtuels,* Biskra, université de Biskra, 2017.

[6] A. Bouguetitiche, *Modilisation des environnements virtuels urbains dédiés à la simulation du mouvement des piétons,* Université de Biskra, 05/10/2020.

[7] C. Briquet, *Algorithme de contournement de barrières,* Université de Liège, Juin 2003.

[8] Arkin, (O) Chenney (S) et Forsyth(D.A)., «Efficient multi-agent path planning,» *Dans: Computer Animation and Simulation 01,* pp. 151-162, 2001.

[9] A. DELLABANI, *Simulation réaliste de Croisement de flux de piétons par intégration de la capture de mouvement,* Université de Biskra, 2012, p. 35.

[10] M. Boucetta, *Planification, navigation et comportements en temps réel d'une foule d'humains virtuels,* Univesité de Biskra, 2012.

[11] Singh, *shawn and Kapadia, Mubassir and Hemeletee, Billy and Reinman, Glenn and Flontsos, Petros ,A modular framwork for adaptive agent based steering,* 2011;symposing on interactive 3D Graphics and games..

[12] Warren, «Multiple robot path coording using artificial potentiel field,» chez *Robotic Rbotic and automation,* C.W 1990.

[13] F. J, *Vers une intelligence collective,* Inter Edition, 1995.

[14] F. Cherif, Animation comportemental : Simulation de fooles d'humains virtuels, Université de Biskra, 2006.

[15] C. Reynolds, «Steering behaviours for autonomous characters,» chez *Proceeding of Game developers conference 1999,* San Fransisco, California, 1999.

[16] B. (R.D), «Intelligence as Adaptive Behavior,» *Experiments in ComputationalNeurothology,* 1990.

[17] Reynolds (C.W), *Flocks,herds and schools : adistributed behavioural model,* Dans : SIGGRAPH'87 21, no 4 of Computer Graphics, 1987, pp. 25-34.

[18] P. (P), *Modélisation 2Ddiscrète dumouvement des piétons : application à l'évacuation des structues du génie civil et à l'interaction foule-passerelle,* Université Paris-Est, 2012.

[19] P. N, *Allbeck(J),Badler (N),Virtual Crowds:Methods,Simulation,andControl.A.B.Barsky,Vol.,* Publication in the Morgan &ClaypoolPublishers series,Synthesislectures on computer graphics and animation,, 2008, p. 176.

[20] D.Helbing, *Buzna(L), Johansson(A.), and. Werner(T.),"Self-organized pedestrian crowd dynamics: experiments, simulations,and design solution, "Transportation Sciencen vol.39,,* 2005, pp. 1-24.

[21] S. Kirchner(A), *Simulation of evacuation processes using a bionics-inspired cellular automaton model for pedestrain dynamics, " Physica A : Statistical Mechanics and its Applications , Vol.312,* 2002, pp. 260-276.

[22] A. (J.), *The evacuation of non-ambulatory patients from hospital and nursing home fires : A framwork for a model. Technical report,* novembre 1997. NBSIR79-1906.

[23] L. Samuel, *Simulation du comportement de suivi dans une foule de piétons à travers l'expérience, l'analye et la modélisation,PHDthesis,* Université de Rennes 1, Avril 2012.

[24] P. S. Kanmeugne, Simulation crédible des déplacements des piéton en temps réeel: modèle microscopique a inuence macroscopique.

[25] D. W, *Anote on two problemsin connexion with graphs, Numerische mathemtik, 1(1),* 1959, pp. 269-271.

[26] A. Bouguetitiche, *Modélisation topologique et sémentique de l'environnement,* Biskra, Mémoire de Magister, Université de Biskra, 2011.

[27] M. Péchaud, «creativecommons,» [En ligne]. Available: http://mpechaud.fr/scripts/parcours/index.html.

[28] M. shakoue, *Conception et implimentation d'un algorithme de planification de chemin dans un jeu video comporant un environnement triangularise,* Université du Quebec a Montréal, Septembre 2012.

[29] S. Riviere, Calculs de visibilité dans un environnement polygonal 2D, Gronoble, Thèse de Doctorat ,Univesité Joseph Fourier Gronoble, 1997.

[30] M.A Wesley, *Lozano-Pérez(T.) , An algorithme for planning collision free paths among polyhedral obstacles.Commun. ACM,vol.22,,* 1979, pp. 560-570..