UQAC
Université du Québec
à Chicoutimi

Génération procédurale de chemins de gardes pour les jeux d'infiltration

Par Audran Bonnot

Mémoire présenté à l'Université du Québec à Chicoutimi en vue de l'obtention du grade de Maître ès sciences (M. Sc.) en informatique.

Québec, Canada

RÉSUMÉ

L'infiltration est une mécanique de jeu qui occupe une place importante dans les jeux. Il s'agit souvent d'une idée simple, aller d'un point A à un point B en évitant le champ de visions de gardes et/ou caméras, gardant une salle avec des obstacles permettant de se cacher. La mécanique a beaucoup grandi au cours des dernières années et on la retrouve maintenant en tant que sous genre dans des grands jeux. Bien que trivial en apparence, la mécanique s'avère être minutieuse dans son développement et l'expérience est primordiale afin de créer des niveaux agréables pour le joueur. Néanmoins, avec l'exploitation de la mécanique, il est possible que des projets soient développés sans cette expérience, résultant en des niveaux pauvrement conçus. Afin de résoudre ce problème, il est intéressant d'analyser les différentes méthodes utilisées dans ces jeux afin de construire un algorithme pouvant lui-même placer les gardes. Pour y arriver, nous analyserons les particularités de la surveillance dans le monde du jeu vidéo. Puis nous étudierons les travaux déjà faits dans ce domaine avant de proposer une nouvelle solution permettant de pouvoir générer procéduralement des gardes dans le cadre d'un jeu d'infiltration.

REMERCIEMENTS

Je tiens tout d'abord à remercier Yannick Francillette et Bob-Antoine Jerry Ménélas pour leur soutient et leur aide tout au long du projet et de la rédaction de ce mémoire. Je tiens également à remercier toute l'équipe du Département d'Informatique et de Mathématique (DIM) de l'UQAC pour leur soutient tout au long de ce parcours. Je tiens enfin à remercier l'administration et le service aux étudiants de l'UQAC pour leur aide durant ces années au Québec.

Je remercie également ma famille pour leur soutien immense et leurs encouragements tout au long de mes études. Mes amis également, aussi bien au Québec qu'en France avec qui j'ai pu rire durant de ces deux années.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

CHAPITRE I – INTRODUCTION

Depuis l'apparition des premières applications informatiques destinées à amuser les utilisateurs vers les années 1950, les jeux vidéo ont beaucoup évolué (Djaouti, 2019). De nos jours, ils constituent l'un des secteurs les plus importants de l'industrie du divertissement car elle dépasse largement en chiffres d'affaires l'industrie cinématographique.

Le jeu vidéo apparait comme structure cohérente qui offre à la fois un défi et une expérience émotionnelle. Pour cela, on comprend que ce système s'adresse avant tout à des **joueurs** qui sont en fait des personnes ayant la volonté d'apprendre à faire des choses dans le but d'éprouver du plaisir. Ainsi, on peut retrouver différents types de joueurs (Alix, 2005). Alors que certains voudront explorer l'environnement présenté, d'autres se concentreront sur les statistiques, performances qu'ils peuvent accomplir dans le jeu. Pour cela, on comprend que tout jeu définira un ou des **objectifs** que devra accomplir le ou les joueurs. Un objectif peut être de construire et ou de détruire un édifice donné, de libérer un prisonnier, d'atteindre un lieu donné, de résoudre une énigme etc. Tout cela, se passe suivant un certain nombre de règles, des **procédures**, qui définissent qui fait quoi, où, quand et comment. Il y aura également un environnement de jeu qui présente le monde, avec les différents objets qu'il contient. On parle souvent de **ressources** pour les objets qui présentent une certaine valeur dans l'environnement de jeu. Il est à noter que le temps est souvent une ressource qui est mis en avant dans les jeux car il est facile à intégrer et à présenter au joueur (Zagal & Mateas, 2010). Bien sûr, un élément intrinsèque à tout jeu est le ou les conflits qui y sont traités. Le conflit permet d'avoir une ligne de démarcation entre les protagonistes. Enfin, un dernier élément qui est essentiel à tout jeu est le degré d'incertitude quant à la réalisation de l'objectif que doit réaliser le joueur. Il convient de noter que l'essence même d'un jeu est dans l'adéquation entre l'attente du joueur et ce degré d'incertitude. De fait, si ce degré

d'incertitude est adapté au profil du joueur alors on parviendra à cet état que l'on qualifie de flow (Sweetser, et al., 2017). À savoir un état de conscience où le joueur sera totalement absorbé par le jeu. Ainsi, si le jeu est trop facile pour le joueur, on peut comprendre qu'il risque d'être ennuyeux alors qu'un jeu trop difficile risque d'engendrer pas mal de frustrations.

Avec tous ces éléments qui entrent dans la composition d'un jeu vidéo, on comprend que les concepteurs, comme cela a été fait dans tous les autres domaines ont voulu créer différents types de jeux. Ce faisant, ils offrent un large éventail de types de jeu au consommateur. Ainsi, on peut citer des jeux d'actions, d'aventure, de tirs, de rôle, de stratégie, de simulation, et autres dont voici une rapide description :

Le jeu d'action est un genre de jeu ayant un **gameplay** demandant au joueur habilité et réflexes afin de réagir à des éléments en temps réel. On parle de gameplay pour décrire l'ensemble des caractéristiques d'un jeu qui caractérisent la manière dont on joue. Dans le cas des jeux d'action, cela peut varier d'un jeu à l'autre. Le genre est très ouvert dans sa description et il est assez commun de le retrouver aux côtés d'autres genres plus précis. *Pong* par exemple, peut être considéré comme jeu d'action car il demande aux joueurs des réflexes afin de renvoyer la balle. Une majorité des jeux d'arcade par la suite peuvent également rentrer dans cette catégorie et l'on comprend alors que le genre enveloppe une grande partie du marché.

Les jeux d'aventure quant à eux représentent en partie l'opposé des jeux d'action. Ici le gameplay est axé sur l'exploration, la narration, les dialogues et la résolution d'énigmes. Une grande majorité de ces jeux ne permettent pas le combat et lorsqu'il y en a, il est souvent scénarisé et demande des mécaniques simples au joueur. Le jeu d'aventure a pour but de raconter une histoire comme un film ou un livre mais se différencie de ces autres supports par son aspect interactif (Dillon, 2005), permettant au joueur d'influencer l'histoire selon ses choix. Encore une fois, même si le genre d'aventure se définit par certains critères, les développeurs ont su le mélanger avec

d'autres genres. Ainsi, le genre action-aventure prend des parties des deux genres, se concentrant sur l'aspect exploration et narration tout en intégrant des combats demandant des bons réflexes au joueur. Il est également possible d'y ajouter d'autres éléments, avec *The Last of Us*[1] qui inclus également des mécaniques de tir ou d'infiltration.

Un jeu de tir est un jeu donnant au joueur la capacité de tirer sur des ennemis ou des objets à l'aide d'une variété d'armes. Bien que simple en apparence, il se divise en un grand nombre de sous catégories. On y retrouve les jeux de tir au pistolet optique qui demandent au joueur de viser avec un objet réel (un pistolet en plastique généralement) afin de tirer dans le jeu à l'endroit viser sur l'écran par le joueur. On peut également citer la catégorie des *Shoot 'em up* dont le fameux *Space Invaders*[2] en est un pilier. Le genre demande au joueur de contrôler un véhicule ou personnage afin de détruire un grand nombre d'ennemis qui apparaissent, le tout en évitant les tirs de ces derniers afin de rester en vie. Enfin, ces jeux de tir sont plus souvent déclinés sous les formes de « *First Person Shooter* » (FPS) ou « *Third Person Shooter* » (TPS), n'incluant pas de règles spécifiques mais un point de vue différent. Les FPS proposent une caméra placée au niveau des yeux du personnage joué alors qu'il sera possible de voir tout le personnage du joueur dans un TPS. Par abus de langage, les termes FPS et TPS se sont retrouvés dans d'autres genres pour décrire le système de caméra. On pourra donc retrouver des jeux « aventure, TPS » alors que ceux-ci ne proposent pas de tir mais juste une caméra à la troisième personne.

Les jeux de rôle proposent généralement un gameplay axé sur l'évolution d'un ou plusieurs personnages au fil d'une quête. Ces jeux contiennent généralement un univers vaste dans lequel le joueur peut s'aventurer librement et un scénario développé avec des quêtes dites secondaires permettant de faire évoluer d'autant plus le

[1] https://www.playstation.com/fr-fr/the-last-of-us/
[2] https://spaceinvaders.jp/index.html

personnage. De ce fait, ils ont généralement une durée de vie élevée. Ils possèdent souvent des éléments appartenant à d'autres types de jeu afin de proposer une grande variété dans le gameplay. Ainsi, il est possible d'avoir des quêtes tournant autour d'une mécanique particulière comme le combat, l'infiltration ou encore les puzzles (Tremblay, Torres, Rikovitch, & Verbrugge, 2013).

Les jeux de stratégie sont centrés sur la planification d'actions tactiques ou stratégiques. Ils présentent souvent un plateau sur lequel le joueur peut faire évoluer des unités afin d'accomplir différents objectifs comme la destruction d'un camp ennemi, la conquête de territoires ou le ralliement des autres unités du jeu. Le genre se sépare en deux grosses catégories. D'un côté les jeux de stratégie en temps réel où le joueur est amené à coordonner tous ces éléments alors que le jeu évolue. Le rythme du jeu est alors influencé par la rapidité pour produire des unités ou la génération des ressources ce qui permet à chaque jeu d'offrir une expérience bien différente. De l'autre côté, les jeux de stratégie en tour par tour proposent un système ordonné avec des actions réalisées en suivant un ensemble de règles. Ils apportent souvent un élément aléatoire dans les affrontements incitant le joueur à définir au mieux sa stratégie afin de réduire l'impact de ce dernier.

Un jeu de simulation est un jeu qui permet au joueur de reproduire une activité dans plusieurs environnements. Ce domaine est extrêmement vaste et il n'est pas rare qu'il croise d'autres genres de jeux. On peut notamment y retrouver des jeux de gestion, des jeux de rôle ou encore des jeux de stratégie (même si ces derniers porteront souvent la nomination de jeu de stratégie plutôt que jeu de simulation de guerre par exemple). De manière plus précise, on peut attribuer le terme de jeu de simulation aux jeux proposant une reproduction d'un environnement avec une grande quantité de détails sur un univers précis. Il est possible par exemple de retrouver des jeux de simulation de conduite de tout type d'appareil avec notamment *Flight Simulator*[3] qui propose une carte

[3] https://www.flightsimulator.com/

du monde détaillée dans lequel le joueur peut prendre le contrôle d'un avion et voler comme il le souhaite. A l'aide de ces éléments, ils sont souvent une bonne source pour les jeux sérieux, qui ont pour but d'éduquer le joueur au sujet choisi (Wood, 2011).

Les jeux d'infiltration sont un type de jeu basé sur une mécanique furtive afin d'éviter des combats souvent difficiles. Dans ces jeux, le joueur doit avancer à travers un environnement délimité (souvent une salle) en évitant les différents obstacles, gardes ou tout outil de repérages afin d'atteindre sa destination finale sans être repéré. Les particularités concernant la mécanique de ces jeux ont beaucoup évolué depuis la sortie de *Metal Gear Solid*[4] en 1998, qui en est un pilier et qui a aidé à en codifier les bases (Stamenković, Jaćević, & Wildfeuer, 2017). L'importance de l'élimination de gardes augmente notamment avec la série des *Hitman*[5] qui propose un scénario axé sur l'utilisation d'armes de poing. Le point de vue du joueur change également pour certains jeux avec Dishonored qui aborde une caméra à la première personne.

Ces mécaniques, qui offraient à l'origine un scénario entier pour un jeu deviennent des portions de jeu. La description des différents genres de jeu nous a montré qu'il était possible et même courant d'intégrer des mécaniques de certains genres dans d'autres. C'est le cas de la mécanique d'infiltration qui peut alors se retrouver dans des jeux qui offrent pourtant un système de combat développé comme dans *Far Cry 3*[6]. Les jeux d'infiltration eux-mêmes se dotent de nouvelles mécaniques appartenant à d'autres catégories de jeu. Ainsi, il n'est pas rare d'avoir accès à des armes de poing afin de neutraliser des gardes ou de pouvoir combattre lorsque le joueur est repéré, s'apparentant alors à un jeu de tir (Tremblay, Torres, & Verbrugge, An algorithmic approach to analyzing combat and stealth games, 2014).

[4] https://www.konami.com/mg/mgs5/
[5] https://ioi.dk/hitman
[6] https://www.ubisoft.com/fr-fr/game/far-cry/far-cry-3

Cette utilisation variée à travers tout type de jeu apporte cependant de nouvelles innovations permettant de renouveler la mécanique. Les technologies toujours plus performantes dans le monde du jeu ont permis aux développeurs d'ajouter de plus en plus de détails. Les gardes sont de plus en plus conscients de leur environnement et leurs comportements sont de plus en plus semblables à ceux des humains. Les gadgets présents sont également de plus en plus futuristes, ce qui permet au joueur de diversifier son approche. On pourra citer parmi ces gadgets les drones de *Splinter Cell: Blacklist*[7] (Figure 1) ou encore les armures invisibles de *Call Of Duty : Advanced Warfare*[8]. Cette mécanique est d'ailleurs revisitée à toutes les époques avec notamment *Assassin's Creed Odyssey*[9] qui propose des mécaniques d'infiltration à l'époque de la Grèce antique, où une puissance divine autorise le joueur à se téléporter entre les gardes. De l'autre côté *Cyberpunk 2077* [10] nous transporte dans un futur proche, rempli d'améliorations cybernétiques et propose une grande liberté d'amélioration au joueur qui pourra choisir de ralentir virtuellement le temps afin de se faufiler entre le regard des gardes ou de pirater les puces de ces derniers afin de les rendre aveugles (Figure 2).

La difficulté dans une phase d'infiltration est orchestrée par une grande quantité de facteurs. Les outils donnés au joueur sont le premier de ces facteurs. Une mini carte par exemple peut comporter plusieurs niveaux de détails, allant de l'affichage basique des obstacles jusqu'à donner le chemin des gardes et le sens dans lequel ils regardent. Les différents gadgets occupent également une grande place dans l'infiltration. Une arme silencieuse permettant de neutraliser les gardes à distance, des brouilleurs de caméras ou encore tout équipement permettant au joueur de se fondre dans l'environnement de manière sonore ou visuel peuvent influencer grandement l'expérience de jeu.

[7] https://www.ubisoft.com/fr-fr/game/splinter-cell/blacklist
[8] https://www.callofduty.com/fr/advancedwarfare
[9] https://www.ubisoft.com/fr-fr/game/assassins-creed/odyssey
[10] https://www.cyberpunk.net/fr/fr/

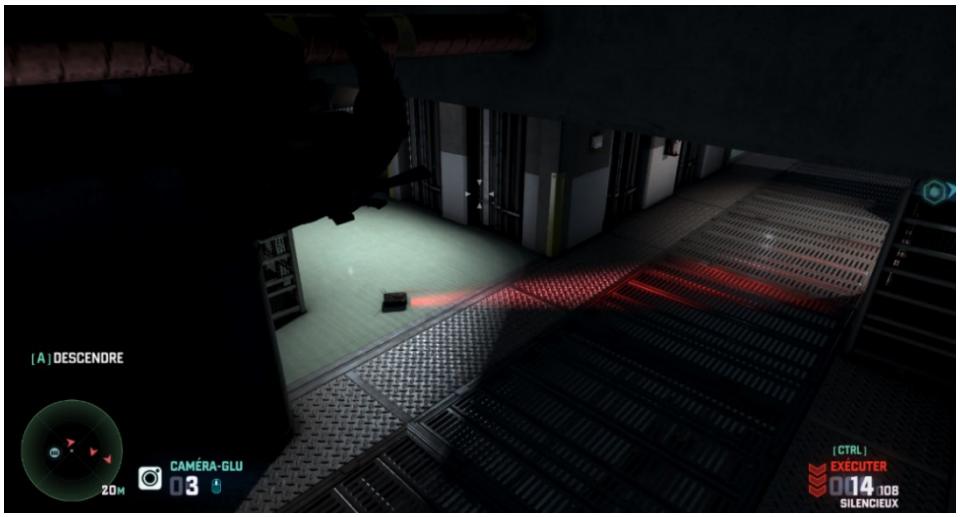

Figure 1 : Drone dans le jeu Splinter Cell : Blacklist *dont on peut voir le champ de vision en rouge*

Figure 2 : Interface de piratage dans le jeu Cyberpunk 2077 *montrant la capacité à aveugler un garde*

Tous ces outils sont adressés directement au joueur afin de lui donner la possibilité de choisir son approche. De l'autre côté, l'intelligence artificielle (IA) dirigeant les gardes est également un facteur important. Considérée comme le cerveau de ces gardes, l'IA leur donne une vision et une ouïe. Certains jeux optent pour une approche permissive à l'égard du joueur, l'autorisant parfois à rentrer dans le champ de vision des gardes sans pour autant l'alerter tout de suite. D'autres en revanche n'auront besoin que d'un bruit léger à côté d'eux pour se mettre en alerte. Ils sont une partie importante dans la transmission de l'amusement au joueur. Des IA trop réalistes et compliquées à battre ne contribuent pas toujours à l'amusement mais il faut néanmoins garder un

comportement humain afin de plaire aux joueurs (Al Enezi & Verbrugge, 2023). Là encore, tout dépend du jeu et des mécaniques proposées, mais il est important de noter que le moteur *Unreal Engine*[11] possède un composant préconstruit pour les IA dans ces phases d'infiltrations, attestant de l'importance accordée à ce type de mécanique dans le monde du jeu vidéo. Le dernier facteur est l'architecture ou de manière générale l'environnement dans lequel évolue le joueur. Le design est souvent réfléchi pour permettre au joueur de se cacher et de contourner les gardes. Certains éléments iront à l'encontre du joueur en revanche, comme du sable qui laisse des traces de pas ou le gravier faisant plus de bruit.

Néanmoins, l'exploitation de la mécanique d'infiltration dans les jeux soulève de nombreux défis, pour parvenir à créer une expérience de jeu amusante. En effet, l'amusement créé par cette mécanique, dépend d'une part de la topologie du niveau de jeu mais surtout des chemins patrouillés par les gardes. Ainsi, le joueur éprouve du plaisir lorsqu'il arrive à trouver une stratégie gagnante pour lui permettre de se faufiler au travers de ces chemins de patrouille, sans se faire repérer (Koster, 2013). Cependant, imaginer des chemins de patrouille intéressants, est quelque chose de très difficile lorsque l'on manque d'expérience. En effet, le concepteur de niveau sans expérience aura besoin de plus de temps et notamment de résultat de test de jeu (« playtest »), qui sont une opération couteuse (Mirza-Babaei, Moosajee, & Drenikow, 2016), afin de se faire une idée de la qualité du niveau proposé. Étant donné que cette mécanique est de plus en plus présente dans différents genres de jeu, certains concepteurs peuvent éprouver de la difficulté à imaginer des niveaux intéressants ce qui peut avoir un impact négatif sur l'expérience de jeu global. C'est pour cela qu'il est intéressant de proposer des outils permettant de calculer des chemins de patrouille pertinents pour les joueurs. Ainsi, la question de recherche de ce travail est la suivante : **Quels sont les métriques**

[11] https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/ArtificialIntelligence/AIPerception/

et approches algorithmiques permettant d'identifier des chemins de patrouille intéressants dans un niveau de jeu donné ?

1.1 CONTRIBUTION DE LA RECHERCHE

La contribution principale de ce travail est une méthode permettant le placement de gardes dans un niveau d'infiltration, en faisant attention à ce que ce dernier soit réalisable en tout temps. Cette méthode propose notamment un placement de garde basé sur le chemin du joueur, une structure particulière de matrice 3D dans lequel évolue tout le programme ainsi que deux méthodes de calcul de difficulté permettant la régulation du programme. Bien que ce sujet puisse être abordé de biens des manières, et notamment via l'aide d'IA, l'approche utilisée ici est assez générique et intuitive, permettant ainsi de servir de base à d'autres travaux similaires si besoin. Il a d'ailleurs été effectué sur *Unity3D* dans la version *2021.3.28*[12] attestant de la simplicité à implémenter et à appliquer à d'autres projets ou travaux si besoin.

1.2 ORGANISATION DU MÉMOIRE

Ce mémoire comporte cinq autres chapitres, dont voici l'organisation :

- Le chapitre 2 explique les différences entre le monde réel et les jeux concernant l'infiltration ainsi que les particularités des éléments de gameplay dans ces jeux.
- Le chapitre 3 présente les critères d'analyses et les méthodes d'autres documents abordant le sujet de l'infiltration.
- Le chapitre 4 montre le développement du projet, ce qui inclus la génération des cartes, des gardes, le contrôle du chemin principal du joueur ainsi que les méthodes de difficulté pour réguler le tout.
- Le chapitre 5 aborde les résultats montrant l'efficacité des différents composants de l'algorithme.

[12] https://unity.com/releases/editor/archive

- Le chapitre 6 est une conclusion basée sur ces résultats ainsi que des voies pour des futures recherches.

CHAPITRE II – GAMEPLAY D'INFILTRATION

Considérant que la surveillance occupe une place centrale dans les jeux d'infiltration, nous analyserons dans cette section la différence entre la surveillance dans la vie réelle et celle que l'on retrouve dans le contexte du jeu vidéo. Ce sera l'occasion pour nous de présenter les points de similitudes et surtout les différences relatives à ces deux situations.

2.1 LA SURVEILLANCE DANS LE CONTEXTE RÉEL

La surveillance dans le contexte réel peut prendre plusieurs formes. Dans le cadre de l'armée par exemple, la surveillance existe pour protéger et pour anticiper une agression. Cette notion de protection et d'anticipation existe également dans la surveillance civile. Depuis les années 2000, les caméras de vidéosurveillance n'ont cessé de prendre une place de plus en plus grande dans le paysage urbain (Haering, Venetianer, & Lipton, 2008). Si le prix d'une caméra est assez faible, le coût humain pour en surveiller les images est en revanche bien plus grand (Ko, 2008). Ainsi, bien que les systèmes d'analyse d'images deviennent de plus en plus performants (Collins, et al., 2000), les caméras seront la plupart du temps utilisées à des fins dissuasives. Une caméra, même aidée par un puissant système d'IA ne pourra faire qu'observer et alerter. C'est également le cas des systèmes d'alarme qui peuvent apparaitre sous plusieurs formes. Dans un magasin par exemple, des antivols sont attachés aux articles sensibles et des portiques placés en entrée et sortie de magasin réagissent à leur présence. Ou de manière plus privée, une alarme peut détecter une intrusion dans une maison à l'aide de capteurs infrarouges. Encore une fois néanmoins, si installer et laisser en marche une alarme n'est pas si couteux, la ressource humaine nécessaire pour agir lorsque cette alarme retentie est beaucoup plus dispendieuse. Ainsi, la surveillance dans le contexte réel cherche à obtenir le maximum d'informations pour un

coût minimum. L’objectif est de dissuader n'importe quelle tentative d'intrusion car le besoin humain qui doit arrêter cette dernière est élevé.

2.2 LA SURVEILLANCE DANS LE CONTEXTE DU JEU

Concernant le monde du jeu, les questions relatives à la gestion d’un budget et les besoins humains sont abstraites. En effet, une fois qu'une caméra ou un garde sont créés et fonctionnels, il est possible d'en placer plusieurs autres quasiment pour le même prix, si l’on considère que ces derniers requièrent peu de ressources logicielles. De plus, la caméra dans le jeu est souvent perçue comme une entité à part capable de repérer le joueur. Le même principe s'applique pour les alarmes et lier ces deux éléments aux gardes présents afin qu'ils réagissent à une intrusion détectée n'augmente que très peu le coût de développement. De ce fait, on comprend que la surveillance dans un jeu pourrait donc être meilleure que dans le contexte réel, voire infaillible.

Néanmoins, les objectifs sont différents. Si le coût n'est pas une contrainte dans les jeux, empêcher l'intrusion n'est pas non plus un objectif. Comme expliqué précédemment, l’objectif n’est pas de frustrer le joueur avec une carte où il serait impossible d’avancer. En fait, afin de parvenir à capter toute l’attention du joueur il convient plutôt, selon le niveau atteint, de lui proposer des scènes présentant une difficulté adaptée. Ainsi, chaque aspect d'une carte d'infiltration, chaque chemin de garde et chaque orientation de caméra seront minutieusement analysés pour laisser au joueur la possibilité d’y passer. On espère parvenir à une bonne adéquation entre les failles du système de surveillance et les habiletés du joueur pour les exploiter. Ainsi, après la complétion d’un niveau, le joueur pourra se glorifier d’avoir été assez habille pour réussir à déjouer le système de surveillance.

2.3 ÉLÉMENTS DE GAMEPLAY EXPLOITÉS DANS LES JEUX D’INFILTRATION

La construction d’un jeu d’infiltration demande de lier une grande quantité d’éléments tout en respectant l’objectif des jeux qui est de procurer du plaisir au joueur. Dans cette

partie nous allons donc présenter les différents éléments retrouvés dans les jeux d'infiltrations qui permettent d'achever un tel objectif.

2.3.1 PLACEMENT DE GARDES SUR DES PARCOURS PRÉDÉFINIS

Le placement des gardes est la première étape afin de définir la surveillance dans un jeu. En regardant son environnement, le joueur doit pouvoir définir un chemin afin de passer entre les gardes. Le placement doit être réfléchi afin de proposer un défi au joueur. Il doit également suivre les objectifs plus haut afin de ne pas bloquer le joueur. Ainsi, en suivant ces objectifs, il est possible d'illustrer ce que représente une surveillance dans un jeu par rapport à ce qui pourrait être fait idéalement si l'objectif était d'empêcher le joueur de passer. En prenant comme exemple une carte inspirée du premier niveau du jeu *Metal Gear Solid* (Figure 3), il est possible de créer des agencements de gardes montrant la surveillance « parfaite » dans un jeu par rapport à celle donnée dans un jeu typique.

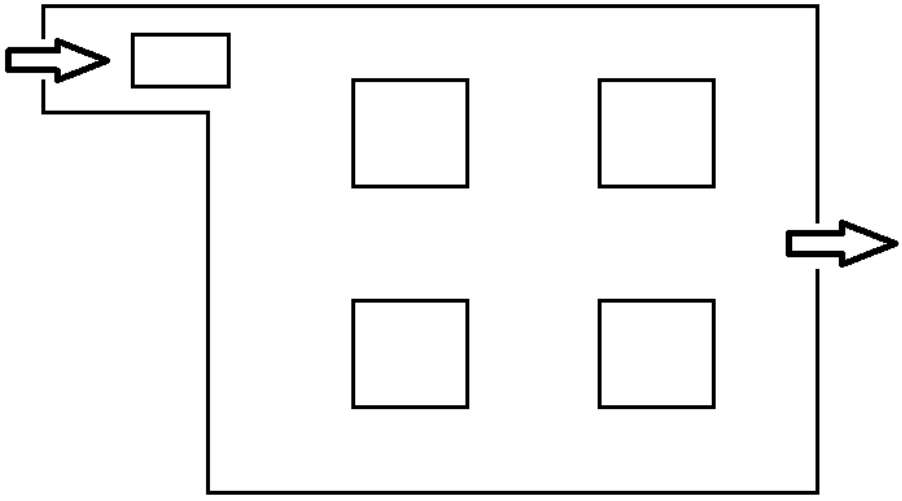

Figure 3 : Illustration de la carte vide inspirée du premier niveau de Metal Gear Solid *avec 5 obstacles*

La Figure 4 représente une surveillance très poussée de la carte. Il est impossible d'imaginer pouvoir passer au travers sans se faire repérer. L'entrée et la sortie sont entièrement bloquées par des gardes et des caméras. La Figure 5 quant à elle représente

un modèle bien plus approchable du côté du joueur. Les gardes couvrent pourtant une zone relativement grande mais ces zones ne sont couvertes que périodiquement via le mouvement des gardes, ce qui laisse des ouvertures pour que le joueur puisse s'infiltrer. La différence entre ces deux illustrations nous montre également comment la difficulté peut varier dans un niveau. En augmentant le nombre de gardes, il devient plus compliqué pour le joueur de retenir les chemins et le risque d'erreurs augmente alors. Mais il est également possible de diminuer l'espace donné au joueur résultant également en une augmentation de la difficulté. Au contraire, des gardes statiques, en petit nombre ou laissant un grand espace sont des facteurs qui facilitent la tâche du joueur.

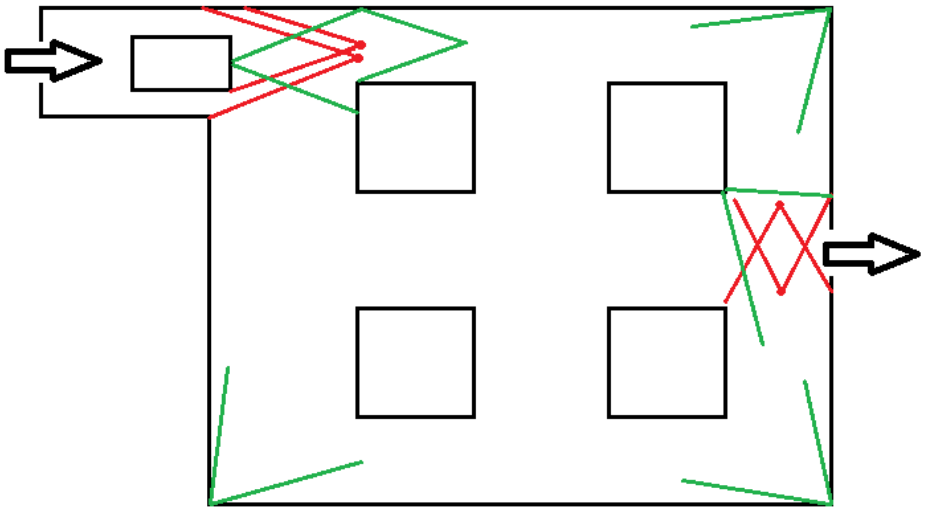

Figure 4 : Illustration d'une surveillance «parfaite». Les champs de vision des gardes sont en rouge tandis que ceux des caméras sont en vert

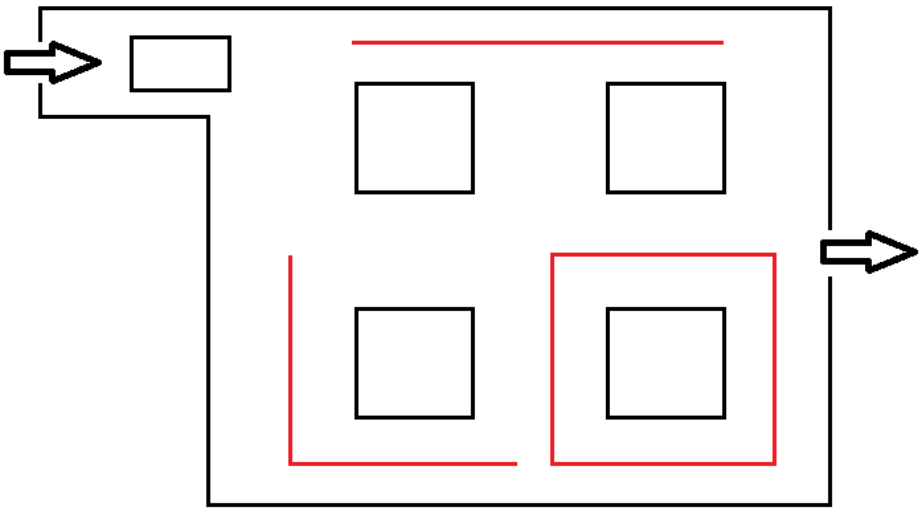

Figure 5 : Illustration de la surveillance type dans un jeu. Les chemins de rondes des gardes sont en rouge

2.3.2 EXPLOITATION DE L'INTELLIGENCE ARTIFICIELLE

Les gardes d'un jeu d'infiltration forment une majeure partie des ennemis rencontrés au cours du jeu. Ils sont également le centre d'attention du joueur car il doit à tout prix éviter de se retrouver dans leurs champs de vision. Mais même une fois placés, ces gardes ne remplissent pas entièrement leurs fonctions et le système d'IA les contrôlant devient alors tout aussi important que le placement. Dans le cadre vidéoludique, l'IA peut par exemple contrôler le déplacement du garde et gérer sa vision, lui permettant de repérer le joueur dans son champ de vision. Cette détection est souvent opérée par la vision mais il existe d'autres sens donnés aux gardes. Le son est devenu tout aussi important, obligeant souvent le joueur à s'accroupir, qui, selon la tradition des jeux, réduit le bruit généré. L'envie de faire un jeu toujours plus détaillé couplé aux évolutions technologiques font que les gardes sont conscients d'une grande quantité d'éléments. Les gardes réagiront à des éléments suspects, tels que des bruits, des traces de sang, un manque de réponses d'un autre garde pendant trop longtemps ou tout autre changement dans l'environnement exécuté par le joueur. Les

gardes deviennent plus perceptifs et conscients du monde qui les entoure. Ils évoluent aussi durant le jeu et en réaction aux actions du joueur le poussant à revisiter sa stratégie entre chaque mission, voire en temps réel. Ils seront également plus astucieux lorsque le joueur est repéré, profitant eux-mêmes des différentes cachettes qui servaient jusque-là au joueur. Ces éléments rendent la phase d'infiltration plus complexe mais aussi plus dangereuse. Malgré ces évolutions, certaines règles dominent. L'IA doit simuler une vision qui parait équitable pour le joueur et le voir à travers les murs n'est certainement pas une option. De même, des chemins aléatoires sont à éviter car cela va à l'encontre de la prévision que le joueur peut faire.

2.3.3 MONDES OUVERTS CONTENANT DIFFÉRENTS TYPES D'INTERACTION AVEC LE JOUEUR

Les gardes ne sont pas le seul élément que le joueur doit étudier. Lors d'une phase d'infiltration, l'architecture de l'environnement est également un grand facteur. Selon les capacités que possède le joueur, l'environnement peut lui permettre de se déplacer différemment ou d'être moins visible. Le jeu *Deus Ex : Human Revolution*[13] permet par exemple de se déplacer d'abris en abris de manière automatique. Cette mécanique aide le joueur à pouvoir mieux planifier son chemin. Si les déplacements entre obstacles sont automatiques, le risque d'imprévu diminue et le joueur peut alors mieux planifier son chemin. Le jeu *Splinter Cell : Blacklist* quant à lui permet au joueur d'éteindre des lumières afin de se déplacer dans l'obscurité. Le jeu rend également cet aspect important en rendant le personnage presque invisible, fortifiant l'idée d'être caché, ne laissant visible que les trois points de son casque pour que le joueur puisse encore se repérer dans le jeu (Figure 6). Les environnements dans lesquels évoluent le joueur sont de plus en plus grands, autorisant

[13] https://www.eidosmontreal.com/fr/jeux/deus-ex-mankind-divided/

plus de liberté et d'interactions. Il est possible de lancer des objets pour attirer l'attention mais cela peut également se retourner contre le joueur qui peut accidentellement faire tomber ces derniers.

Figure 6 : Capture d'écran du jeu Splinter Cell : Blacklist. *On peut y voir en haut trois points bleus représentant le casque de vision nocturne du personnage joué.*

2.3.4 VARIATION DE DIFFICULTÉ

La difficulté dans un jeu d'infiltration est altérée en grande partie par les éléments vus précédemment. Ainsi, une plus grande quantité de gardes résulte souvent en une augmentation de la difficulté. Le joueur doit observer plus longtemps les rondes, les zones couvertes par les gardes sont plus grandes et les risques d'imprévus sont donc augmentés (Smith, 2006). Un jeu peut aussi changer les sens des gardes, les affaiblissant volontairement au début du jeu pour laisser au joueur le temps de s'habituer aux commandes. Il est possible d'apporter une justification à ce changement en introduisant des gardes « plus performants », signe que le joueur peut s'attendre à être plus facilement repéré. Certains jeux peuvent également choisir de faire ressentir les choix du joueur dans

les niveaux. Il est possible que le nombre de gardes et leur vigilance soient affectées par des niveaux précédents. Ainsi, si le joueur exécute des infiltrations parfaites tout au long du jeu, le scénario peut proposer un niveau moins bien gardé car le camp ennemi n'est pas conscient de sa présence. En revanche, si le joueur est vu à chaque niveau, la réaction logique serait de mieux garder d'autres lieux, menant à des niveaux plus protégés. Malgré ces éléments, il se peut que certains joueurs optent pour une stratégie visant à augmenter le nombre de gardes. Ces variations de difficultés entre les niveaux peuvent également être inversées et rendre le jeu plus difficile si le joueur réussit est aussi une approche valable. Un niveau difficile dans un jeu d'infiltration implique certes une plus grande réflexion mais également une plus grande satisfaction lorsque celui-ci est réalisé avec perfection. Ainsi, différents niveaux de difficulté seront proposés au joueur (souvent au début qui est ensuite paramétrable durant le jeu) afin d'adapter les gardes et leurs comportements à tout type de joueur.

2.4 EXPLORATIONS DE DIFFERENTS SUPPORTS

Le support dans lequel le jeu est apporté est également un aspect important dans la transmission de l'amusement. Même si les jeux à grand budget sont généralement prévus pour consoles et ordinateurs, il arrive que certains soient exclusifs à un format en particulier. Jusque récemment, le support ne changeait que peu la manière de jouer à un jeu car il est courant d'admettre qu'une manette ou un clavier peut être utilisé dans chaque catégorie (à l'exception de quelques consoles comme la *Wii*[14] qui fonctionnent grâce aux capteurs de la manette). Néanmoins depuis les années 2010 (Coburn, Freeman, & Salmon, 2017), les casques de réalité virtuelle deviennent de plus en plus accessibles et de nouveaux jeux voient le jour sous ce support. Les jeux d'infiltrations font partie de cette branche de

[14] https://www.nintendo.fr/Wii/Wii-94559.html

nouveaux jeux avec notamment *Espire 1 :VR Operative*[15] ou même *Hitman 3* qui, en plus de sortir en version classique sur consoles et ordinateurs, fonctionne avec le *PlayStation VR*[16].

La réalité virtuelle transporte le joueur plus près du jeu que n'importe quel autre support. A l'aide d'un casque, l'affichage du jeu est juste devant les yeux du joueur et légèrement différencié entre la droite et la gauche afin de compléter l'immersion. Le casque est également équipé de capteurs et le mouvement de la tête est retranscrit dans le jeu. De même, deux manettes permettent de placer les mains du joueur dans le jeu. Certains supports vont plus loin et offrent même des tapis afin de se déplacer, poussant l'immersion au maximum. Même si les objectifs des jeux restent les mêmes, le changement de point de vue et de support change également les règles de jeu (Pallavicini, Pepe, & Minissi, 2019). Le joueur est considéré comme vivant dans le monde et il faut donc adapter les mécaniques déjà existantes afin de garder le jeu amusant. Les sens du joueur sont constamment mis à l'épreuve et le jeu doit rester agréable aussi bien dans sa jouabilité que dans sa transmission d'informations afin d'éviter les troubles dus à la sensation de mouvement. L'architecture doit être repensée par endroits car il ne serait pas agréable pour le joueur de se retrouver la tête à l'envers par exemple.

[15] https://espire1.com/
[16] https://www.playstation.com/fr-fr/ps-vr/

CHAPITRE III – ÉTAT DE L'ART

3.1 CRITÈRES D'INCLUSION

Le sujet des jeux d'infiltration n'est pas nouveau et la génération des gardes est un sujet qui a déjà été étudié. Pour l'analyse de l'état de l'art, nous avons sélectionné tous les travaux publiés depuis 2018, disponibles sur les bases de données IEEE Xplore, ACM Digital Library, Springer. Nous avons utilisé les mots clefs : « *stealth games* ». Cela n'a pas donné une grande quantité de résultats. Ainsi nous avons décidé d'étendre la date de publication des travaux à 2013. Les résultats étaient toujours peu nombreux mais en recherchant les différents travaux des auteurs, une multitude de travaux sur le sujet sont ressortis.

3.2 CRITÈRES D'ANALYSE

Le problème de l'utilisation de la mécanique d'infiltration dans les jeux peut se décomposer en trois sous problèmes qui doivent généralement être traités par le concepteur de niveau. Ces sous problèmes sont : la division de l'environnement en zone « contrôlées », l'évolution de ces dernières au fils du temps et la difficulté pour le joueur à trouver la « faille » de ce contrôle. Ces critères sont définis dans les sous sections suivantes.

3.2.1 DIVISION DE L'ESPACE

L'objectif des entités de surveillance est de contrôler des zones. C'est-à-dire détecter si un intrus (le joueur) se trouve dans certains espaces. Cette détection amène à l'échec de l'infiltration pour le joueur. Ainsi, une question à laquelle le concept de niveau est confronté consiste à définir des zones de contrôle. Cela se matérialise par le choix (dans le cas ou plusieurs entités ayant des caractéristiques différentes existent dans le jeu) et le positionnement des entités de surveillance dans l'environnement. Ainsi, il se trouve découpé en plusieurs zones chacune contrôlée pour un ensemble d'entités qui peut être vide. C'est-

à-dire qu'aucune entité ne contrôle ce sous espace. Nous aborderons ce dernier point dans la sous-section 3.2.3. La Figure 7 par exemple, illustre une surveillance de deux zones importantes autour de l'obstacle. Ici la division est faite de manière à laisser deux grandes zones non contrôlées mais en gardant les points de passages autour de l'obstacle. Finalement cette question peut se résumer en **celle du choix et du positionnement des gardes**.

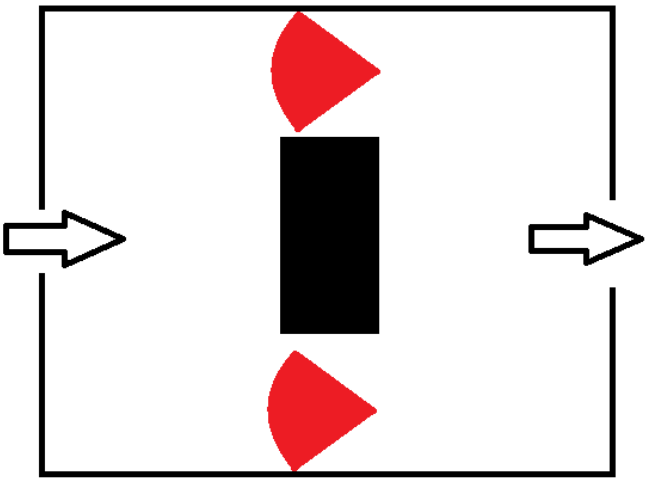

Figure 7 : Illustration de la zone contrôlée par la surveillance de deux gardes (rouge) autour d'un obstacle

3.2.2 ASPECT TEMPOREL

L'autre question se situe au niveau de l'évolution de ces espaces contrôlés dans le temps. L'un des aspects de la mécanique d'infiltration est de modifier ces zones au cours du temps. Cela permet notamment de créer une dynamique qui oblige le joueur à se déplacer, explorer et planifier l'infiltration. Cela est un élément clef du plaisir de jeu, dans cette catégorie de mécanique. Par exemple, les gardes positionnés sur plusieurs points, vont tourner la tête (qui représente le contrôle de zone) à une certaine fréquence, ce qui oblige le joueur à comprendre ce schéma afin de ne pas se faire détecter. Avec cet exemple simple, nous avons une image de comment ces zones peuvent évoluer dynamiquement. Un autre exemple serait de considérer que les gardes peuvent se déplacer. La Figure 8 illustre

ces déplacements. En faisant bouger chaque garde sur sa ligne et en retournant leurs champs de vision de 180° à chaque extrémité, plusieurs zones qui n'étaient alors pas couvertes le deviennent périodiquement, ce qui modifie la manière dont le joueur doit s'infiltrer. Ainsi, le cœur de cette question est d'évaluer la pertinence en termes de plaisir de jeu, de l'influence qu'à cette évolution temporelle sur les zones de contrôle. Cette question peut se résumer **en celle du choix des chemins de patrouille**.

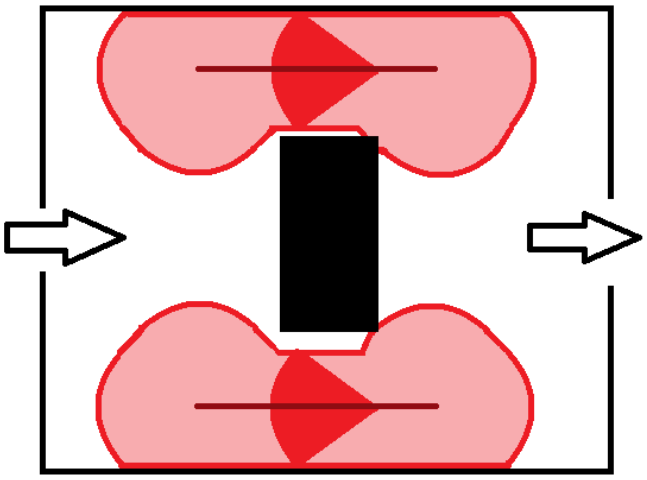

Figure 8 : Illustration de la zone contrôlée par deux gardes (rouge) et leurs chemins de ronde (rouge pourpre)

3.2.3 DIFFICULTÉ DE L'EXPLORATION

Enfin, la dernière question, se situe au niveau de la difficulté créé par la dynamique des deux éléments présentés précédemment. Comme nous l'avons mentionné dans la sous-section 3.2.1, il se peut que certaines zones ne soient contrôlées par aucun garde. Cet aspect est très important car c'est par ces zones, que le joueur doit réaliser l'exploration de la zone de jeu et donc l'infiltration. Plus l'espace non contrôlé sera petit, et plus le niveau de jeu sera difficile. Car cela signifie que le joueur ne dispose que de peu de zone pour évoluer. Ainsi, il est possible de rajouter un peu de chemin aux gardes présents sur la Figure 8 afin d'obtenir une plus grande surveillance de zone. La Figure 9 représente cette extension de

chemin et on peut ainsi voir que la zone contrôlée est plus grande que lorsque les gardes étaient immobiles (Figure 7). Néanmoins, cette couverture plus grande est également une faille permettant au joueur de passer. En effet, s'il est compliqué pour le joueur de contourner l'obstacle sans être vu dans la Figure 7, il sera plus simple de passer sur la Figure 9 lorsqu'un ou les deux gardes sont au bout de leurs chemins respectifs. Comme nous l'avons vu dans le chapitre 2, c'est une des différences entre le monde réel et celui du jeu. Ainsi, même si le placement des gardes et la définition de leurs chemins peuvent être effectuées indépendamment, une faille doit exister pour le joueur. Cette question peut se résumer en celle **de la difficulté du niveau en fonction du choix des gardes et de leur chemin de patrouille**.

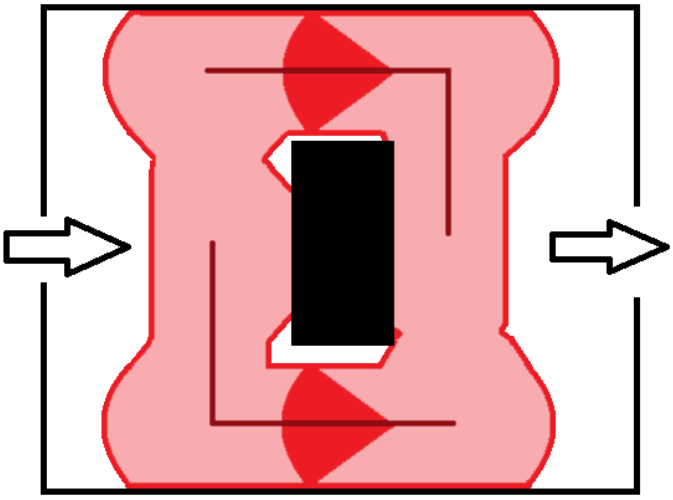

Figure 9 : Illustration de la zone contrôlée par deux gardes (rouge) et de leurs chemins de rondes étendus (rouge pourpre)

3.3 REVUE DE LITTERATURE

Dans cette partie, nous allons analyser les résultats jugés pertinents des recherches selon les trois critères définis précédemment.

3.3.1 APPROCHE DE PLACEMENT PROCEDURAL DE GARDES DANS LES JEUX D'INFILTRATION

Le premier travail de recherche propose une solution pour la génération procédurale de gardes (Xu, Tremblay, & Verbrugge, Procedural guard placement for stealth games, 2014) et propose une découpe de l'espace à l'aide de régions de Voronoi. Un diagramme de Voronoi est une méthode permettant de découper le plan en régions à l'aide d'un certain nombre de points, appelés « graines ». Chaque graine va ensuite se voir attribuer une zone dans laquelle chaque point sera plus proche de la graine que de n'importe quelle autre dans le plan (Figure 10).

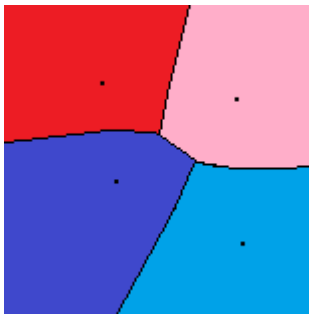

Figure 10: Illustration de quatre régions de Voronoi avec leurs graines

Les gardes se voient attribuer une zone de Voronoi chacun et leurs chemins sont définis de manière à être linéaire à l'intérieur de cette même région. Un algorithme de résolution de jeux d'infiltration (Tremblay, Torres, Rikovitch, & Verbrugge, 2013) est ensuite lancé un certain nombre de fois afin d'obtenir la probabilité de trouver un chemin selon le nombre de gardes. La méthode est intéressante mais les résultats impliquent qu'il n'y aura pas forcément toujours un chemin pour le joueur, ce qui est un critère primordial pour nous.

3.3.2 APPROCHE DE GÉNÉRATION DE GARDES ET CAMÉRAS DANS LES JEUX D'INFILTRATION

Cet article (Xu, Tremblay, & Verbrugge, Generative methods for guard and camera placement in stealth games, 2014) est semblable au premier aussi bien par son nom que par ses auteurs. Ici néanmoins, une méthode de squelettisation est utilisée afin de diviser l'espace entre les obstacles. La squelettisation désigne de manière globale tous les algorithmes permettant d'analyser la forme d'un objet et d'en créer un squelette intérieur comme montré sur la Figure 11. S'il est possible de trouver le squelette de n'importe quel polygone (Attali, 1995), il est plus compliqué de le faire dans le cadre d'une carte avec obstacles car il s'agit d'un polygone concave (à trous). Même s'il existe des méthodes plus complexes afin de résoudre ce problème (Hoffmann, Kaufmann, & Kriegel, 1991), l'approche de considérer la carte comme si un polygone simple a été choisi.

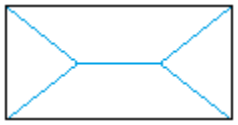

Figure 11: Illustration de la squelettisation d'un rectangle

Une fois le squelette créé, il est approximé puis simplifié afin d'obtenir des segments. Les gardes sont placés sur ces segments et les suivent tout au long de la patrouille entre deux points *a* et *b* choisis. Cette méthode permet non seulement de diviser l'espace mais également de créer des chemins complets pour les gardes. Pour finir, un algorithme « *Rapidly exploring Random Tree* » (RRT) est utilisé afin de trouver le chemin du joueur à travers le niveau. Là encore, les résultats abordent la probabilité de l'algorithme à finir le niveau, sans jamais le garantir, ce qui ne correspond pas à notre troisième critère.

3.3.3 RECHERCHE DE CHEMINS A L'AIDE DE LA SQUELETTISATION DANS UN JEU D'INFILTRATION

Ce troisième travail de recherche propose une solution de chemin pour le joueur à l'aide d'un algorithme de squelettisation (Singh, 2015). De même que pour le travail précèdent (Xu, Tremblay, & Verbrugge, Generative methods for guard and camera placement in stealth games, 2014), l'algorithme va obtenir un squelette passant au milieu des obstacles. Cette fois ci néanmoins, le champ de vision de chaque garde est considéré comme un obstacle. L'aspect temporel est mis au premier plan et le champ de vision des gardes est projeté en trois dimensions avec le temps comme troisième dimension. On peut donc voir l'évolution des chemins pendant leurs rondes dans le temps en tant que structure et non pas seulement en tant que mouvement. L'algorithme va créer un squelette du polyèdre formé et il est possible à partir de ce squelette de former un chemin pour le joueur. Les résultats comparent cet algorithme avec un algorithme RRT sur différents critères dont la taille du chemin et le temps nécessaire pour le trouver. Ces métriques peuvent être jugées importantes dans le cas d'un jeu d'infiltration et se feront ressentir lors de la comparaison avec un algorithme aléatoire comme le RRT. Néanmoins, cela ne fait pas partie de nos critères.

3.3.4 APPROCHE POUR MESURER LE RISQUE DANS LES JEUX D'INFILTRATION

Ce dernier document aborde uniquement la notion de risque dans les jeux d'infiltration. Il nous présente trois formules afin d'obtenir une notion de difficulté et les compare (Tremblay, Torres, & Verbrugge, Measuring risk in stealth games., 2014). La première méthode est un calcul basé sur la distance. Il est d'intuition de penser que plus on se rapproche d'un garde, plus les risques d'être repéré sont grands. Cette méthode se concentre donc sur la distance entre le joueur sur son chemin pour chaque intervalle de temps t et tous les gardes présents sur la carte. La distance est calculée via un algorithme

*A** afin de prendre les obstacles en compte. La seconde méthode testée est appelée « *Line Of Sight* » (LOS). Cette méthode prend en compte tous les aspects du champ de vision de chaque garde tout en gardant l'aspect de distance. Elle élimine également de l'équation les gardes qui n'ont pas une ligne de mire directe avec le joueur. Il y a ensuite un calcul avec l'angle entre le joueur et le sens des gardes, tout en comptant la distance entre les deux. La dernière méthode est appelée « Near Misses » (NM) et prend en compte le chemin du joueur dans son intégralité afin de mesurer la proximité entre différents points dans le temps par rapport aux champs de vision des gardes. Par exemple, pour une étape *t*, la difficulté va également prendre en compte la proximité avec des gardes sur l'étape *t-1*, *t-2*, *t+1*, et *t+2*.

L'analyse de ces algorithmes s'avère assez complexe car ils retournent des valeurs qui ne sont pas interprétables entre elles. Les recherches se concentrent ici sur plusieurs niveaux, chacun possédant des gardes et deux chemins différents pour le joueur. Chaque algorithme va calculer sa valeur de difficulté pour chaque chemin, ce qui permet d'établir un ordre de difficulté par algorithme pour chacun des chemins. Un groupe d'humain a également classé ces mêmes chemins et on peut alors comparer les résultats des algorithmes. En moyenne, il semblerait que l'algorithme de distance est celui qui donne les résultats les plus « humains ». L'algorithme LOS donne également des bons résultats et n'est pas à éloigner pour autant. Enfin l'algorithme NM semble être dernier et donne des résultats plus éloignés. La deuxième analyse appuie cette idée en montrant le pique de difficulté dans chaque niveau pour chaque algorithme et les deux premiers donnent des résultats similaires tandis que NM place le pique de difficulté à un autre endroit.

3.4 DISCUSSION

Dans cette section, nous allons discuter des différents travaux de la littérature en fonction des trois critères présentés. Le Tableau 1 présente un résumé de l'analyse de ces travaux en fonction de ces critères.

Tableau 1 : Tableau de comparaison des différentes approches selon les critères

Approche	**Division de l'espace**	**Aspect temporel**	**Difficulté de l'exploration**
(Xu, Tremblay, & Verbrugge, Procedural guard placement for stealth games, 2014)	Division de l'espace à l'aide de régions de Voronoi.	Chemins définis pour être des lignes dans les régions de Voronoi.	Incomplet car l'algorithme pour le chemin du joueur ne donne pas forcément un chemin.
(Xu, Tremblay, & Verbrugge, Generative methods for guard and camera placement in stealth games, 2014)	Algorithme de squelettisation afin de créer une suite de lignes contournant les obstacles.	Les chemins sont définis comme étant les lignes créés par l'algorithme de squelettisation.	Incomplet car l'algorithme pour le joueur ne donne pas forcément un chemin.
(Singh, 2015)	Non applicable	Les chemins ne sont pas définis par un algorithme mais la projection en trois dimensions donne un aspect physique à l'aspect temporel.	L'algorithme donne un chemin lorsque celui-ci existe mais il n'y a aucune métrique indiquant le niveau de difficulté.
(Tremblay, Torres, & Verbrugge, Measuring risk in stealth games., 2014)	Non applicable	Non applicable	Les chemins du joueur sont définis par un humain et existent donc. De plus trois métriques sont proposés afin d'évaluer la difficulté d'un niveau.

Si on analyse le tableau sur le critère de la division de l'espace, nous pouvons voir que seul Xu, et al. ont étudié cette question et proposent deux approches. L'une utilisant les

régions de Voronoï, et l'autre un algorithme de squelettisation. La méthode des régions de Voronoï a l'avantage d'être flexible. Il est possible de modifier la quantité de zones car peu de paramètres sont requis. En revanche, la génération des graines est généralement aléatoire pouvant ainsi générer des zones qui ne seraient pas bien adaptées à toute les cartes. La méthode de squelettisation quant à elle, s'adapte bien mieux, et a pour but de contourner les obstacles, au prix de la flexibilité.

En analysant le critère temporel, on peut observer qu'il est étroitement lié à celui de la division de l'espace. La définition de chemin dans le premier de ces travaux est linéaire mais respecte la limite des régions de Voronoï définit précédemment. Le placement souffre donc du même défaut d'adaptabilité que la méthode de division de l'espace. La forme linaire est standard et ne permet pas forcément de suivre les formes des obstacles ou de la carte. La méthode de placement utilisée via la squelettisation est en revanche très intéressante. Les lignes formées par l'algorithme servent également de chemins pour les gardes. Ceux-ci suivent donc bien la forme de la carte et peuvent contourner les obstacles. Enfin sur le Tableau 1, on peut remarquer que le travail de Singh rempli en partie le critère temporel malgré le manque de division de l'espace. Bien que ce travail ne se concentre pas sur la génération de gardes (et donc de leurs chemins), il propose là une méthode intéressante d'interpréter le champ de vision des gardes dans le temps. Cette méthode peut par ailleurs être appliquée sur d'autres travaux et ne modifie la définition des chemins.

Enfin, si on analyse le critère de la difficulté, on peut remarquer qu'aucun ne résout le problème dans sa totalité. Les travaux de Xu, et al. n'incluent pas la certitude d'un chemin pour le joueur. Les résultats montrent des probabilités de trouver un chemin en fonction du nombre d'entités sans jamais en assurer la certitude. Cette approche permet d'une manière d'analyser l'efficacité du placement des gardes et retourne une valeur de difficulté. Plus la probabilité est faible, plus le niveau peut être considéré comme étant difficile. Le travail de

Singh se concentre exclusivement sur le chemin du joueur et l'algorithme obtient donc un résultat du moment que ce chemin existe. En revanche, aucune métrique indiquant la difficulté ne peut vraiment ressortir. Enfin, Tremblay, et al. ont spécialement étudié la question de la difficulté et le calcul des risques dans le cadre des jeux d'infiltrations. Même si leur approche ne propose pas de chemins créés par algorithme, les trois méthodes proposées peuvent être appliquées de manière générale et représenter des métriques de difficulté dans tous les cas de phase d'infiltration.

Le Tableau 1 nous montre finalement qu'aucun des travaux ne respecte entièrement les trois critères. Les deux premiers travaux sont néanmoins assez proches et abordent bien le sujet de la génération procédurale de garde mais omettent partiellement l'aspect de faisabilité, qui est primordial dans le cadre d'une phase d'infiltration. Sur ces deux approches, l'aspect temporel est lié à la division de l'espace même s'il est plus trivial dans le cas des régions de Voronoi. Néanmoins, la troisième approche propose une méthode pour interpréter les chemins de garde dans le contexte d'un jeu. Bien qu'il ne s'agisse pas directement de la définition de ces chemins, cette méthode apparait dans l'aspect temporel. Enfin, le critère de difficulté n'est jamais totalement complet. Les résultats montrent des probabilités de trouver un chemin et lorsque le travail tourne autour du chemin, les métriques sont manquantes. Le dernier travail possède l'avantage d'analyser plusieurs métriques qui peuvent être utilisées de manière générale dans le cadre d'une phase d'infiltration.

3.5 CONCLUSION

L'analyse de la littérature existante sur le sujet a montré que même si le sujet de la génération de gardes dans un jeu d'infiltration a été abordé, il manque certains éléments par rapport à nos critères. Le Tableau 1 nous montre d'ailleurs que seuls deux documents ont abordé le sujet de la génération de gardes dans un jeu d'infiltration pur, c'est-à-dire, sans gadget ou ajout d'autres éléments (distraction par exemple). Ces deux documents

présentent deux méthodes de division de l'espace et deux méthodes de définir des chemins pour les gardes. Le travail de Singh quant à lui aborde le sujet du côté du joueur et essaie de trouver un chemin à l'aide d'un algorithme de squelettisation. L'approche en trois dimensions du projet peut être appliquée pour tout autre projet et est une idée intéressante afin d'interpréter le mouvement des gardes dans le temps. Enfin le travail de Tremblay, et al. présente des métriques intéressantes qui ont été comparées au comportement humain et qui sont applicables à tout type de projet d'infiltration.

CHAPITRE IV – RÉALISATION

Ce chapitre décrit l'approche proposée afin de répondre au problème de recherche. Il est divisé en quatre sections. La première présente la génération des niveaux sur lesquels l'algorithme va évoluer ainsi que quelques détails sur le fonctionnement des gardes. La deuxième explique comment le chemin du joueur est généré avec l'algorithme A* et dans quelle structure il évolue. La troisième décrit la manière de placer les gardes sur la carte ainsi que la définition de leurs chemins dans le niveau. La dernière partie quant à elle présente deux formules de difficultés utilisées afin de contrôler le placement de gardes. Elle aborde également plus en détails le fonctionnement de la boucle de l'algorithme.

4.1 CRÉATION DU PROJET ET INITIALISATION DES GARDES

Notre approche prend pour entrée une topologie de niveau défini par les concepteurs de jeu. Dans cette recherche, afin d'avoir un banc de test permettant d'appliquer et d'évaluer notre approche, nous simulons un niveau créé par quelqu'un sans expérience.

4.1.1 GÉNÉRATION PROCEDURALE D'UNE CARTE

La génération de la carte doit suivre le raisonnement simple d'une personne n'ayant aucune expérience dans le domaine du design de carte. Ainsi, il est possible de générer une bonne partie des valeurs aléatoirement. Les dimensions de la carte, par exemple sont générées de manière aléatoire entre deux bornes aussi bien pour la largeur et la longueur. Le nombre d'obstacles est ensuite calculé en fonction de la taille de la carte et ils sont placés aléatoirement sur la carte. Les résultats ne sont pas tous adaptés au projet car trop pauvres en diversité mais certaines générations donnent des résultats exploitables. Ainsi, plus de cinq cents configurations de carte ont été testées et les six meilleures ont été retenues pour

les tests. Il s'agit de cartes pas trop grandes avec des obstacles qui apportent un aspect intéressant. Le pseudo code de cette création peut être trouvé sur la Figure 12.

Algorithm 1 Map Generation

```
1:  procedure CreateMap(seed)
2:      Random.InitState(seed)
3:      width ←Random(10, 30)
4:      length ←Random(20, 50)
5:      nbObstacles ← width ∗ length/150
6:      for i in 0..nbObstacles do
7:          scale ←Vector2(Random(1, 7), Random(1, 7))
8:          pos_x =Random(−length/2 + scale.x, length/2 − scale.x)
9:          pos_y =Random(−width/2 + scale.y, width/2 − scale.y)
10:         CreateObstacle(pos_x, pos_y, scale)
11:     end for
12: end procedure
```

Figure 12 : Pseudocode de la fonction permettant la création des cartes

4.1.2 GÉNÉRATION DE LA MATRICE

La création de la carte place également deux points, dans deux coins opposés de la carte. A partir de ces deux points, il est possible de créer une matrice de points centrée au milieu de la carte et s'étendant jusqu'aux murs. Chaque point de cette matrice possède une sphère de collision qui va interagir avec les différents éléments de la carte. Ici les cartes sont des rectangles donc à part les obstacles, il n'y a pas beaucoup d'interactions particulières mais le système fonctionne également sur toute forme en reconnaissant le sol (et donc le vide).

On peut voir sur la Figure 13 que les points à droite de l'obstacle sont rouges, reflétant de la présence de l'obstacle. L'avantage de ce système est sa simplicité aussi bien dans le développement que dans l'exécution. Il est également flexible et l'espacement entre chaque point est défini par la distance que le joueur peut parcourir entre chaque intervalle de temps.

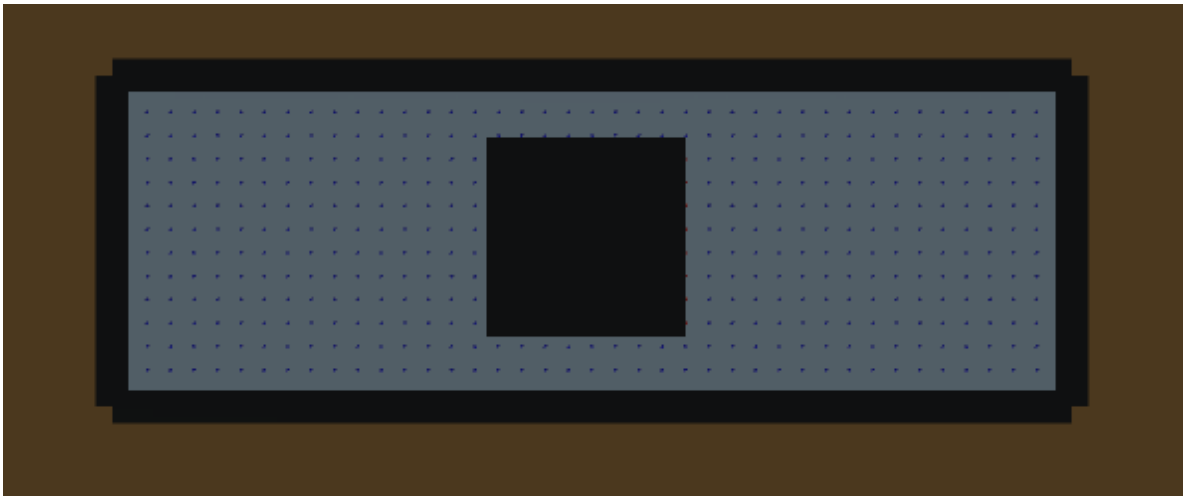

Figure 13: Capture d'écran montrant l'interaction entre la matrice et un obstacle

4.1.3 CRÉATION DU MODÈLE DES GARDES

Les gardent doivent également être un composant simple et modulable du projet. Ici ils sont représentés par un point, un angle de vu, une distance de vu et un *mesh* qui représente leurs champs de vision. Le *mesh* n'est pas obligatoire mais il permet d'avoir un retour visuel lors des tests ce qui a été d'une grande aide tout au long du projet. Les gardes, lors de la construction de leurs champs de vision vont interagir avec la matrice de la carte pour changer les informations de cette dernière. Les points de la matrice compris dans le champ de vision vont alors être considérés comme vus par un garde.

On peut observer en regardant de près sur la Figure 14 que les points de la matrice interagissent avec le garde. Le champ de vision est tracé par rayon depuis le centre et chaque point rencontré va changer d'état. Même s'il est rouge comme pour les obstacles, son « tag » est différent car il ne va influencer que le chemin du joueur alors que les obstacles comptent comme des zones inaccessibles pour toutes les entités.

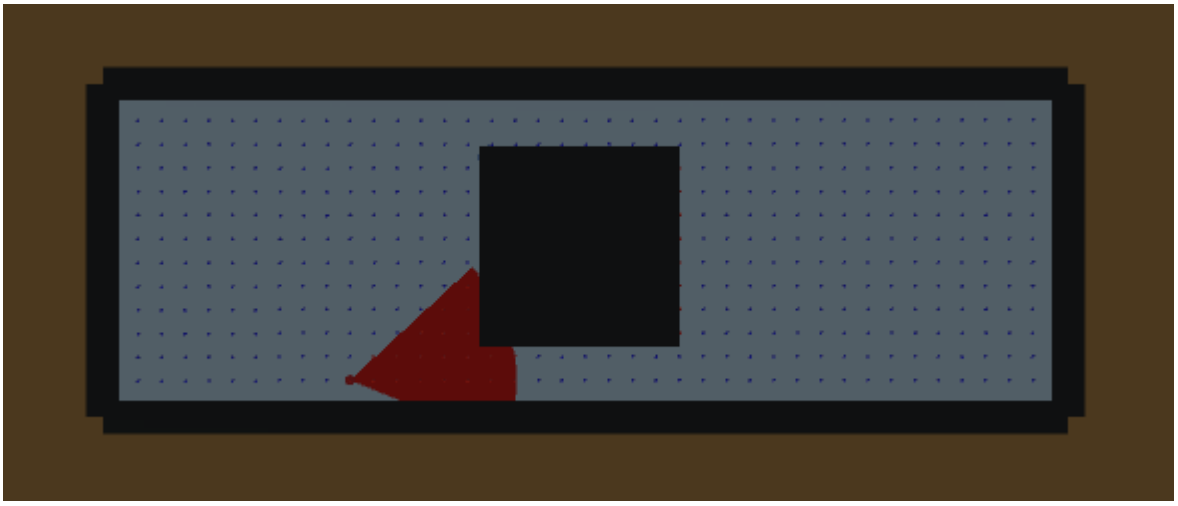

Figure 14: Capture d'écran montrant un garde et son champ de vision

4.2 DÉFINITION DU CHEMIN DU JOUEUR

Dans le domaine de la recherche de chemins, beaucoup d'algorithmes existent. Néanmoins, l'analyse de la littérature nous montre que certains ne donnent pas forcément un chemin. Ainsi il est possible de choisir un algorithme A*. Même si celui-ci possède en général des performances moindres que d'autres algorithmes (Delling, Sanders, Schultes, & Wagner, 2009), il peut encore être le meilleur dans certains cas (Zeng & Church, 2009). De plus, l'algorithme A* va trouver un chemin tant que ce dernier existe (Russell, 2010) ce qui permet de satisfaire notre troisième critère. Il ne nécessite ensuite qu'un réseau de nœuds (points reliés) qui peut être issue de n'importe quelle structure notamment une carte projetée en 3D. L'algorithme fonctionne normalement en regardant le coût pour passer d'un nœud à un autre. Dans notre cas, ce dernier peut être standard et définit à n'importe quelle valeur du moment qu'il reste constant pour tous les nœuds. En effet la matrice est construite de manière à ce que les points soient espacés d'une certaine distance, mais cette distance ne se transmet pas vraiment dans l'aspect en trois dimensions de la matrice virtuelle. Ainsi il est possible de choisir n'importe quelle valeur (ici 1). Pour chaque déplacement de l'algorithme dans l'espace, il se déplacera également d'une unité dans le temps, ce qui

correspond dans la matrice à un ajout de couche. A chaque ajout de cette couche, l'état des points sur la carte sera mis à jour pour refléter en direct les changements sur la carte. L'algorithme A* qui trouve le chemin se déplacera alors d'une unité en diagonale maximum entre chaque couche.

Le chemin sur la Figure 15 est construit avec le point le plus en bas à gauche en tant que départ et le point le plus en haut à droite en tant qu'arrivée. Il ne reflète pas exactement le chemin qu'un joueur pourrait prendre mais il nous assure qu'il est possible de finir le niveau. Comme l'algorithme ajoute des couches dans la matrice au besoin, il faut aussi le limiter à ce niveau pour ne pas finir dans une boucle infinie. Ici la limite est deux fois la taille du chemin actuel, cette limite est aussi pensée en accord avec l'expérience utilisateur car un chemin trop complexe peut signifier mauvais niveau d'infiltration.

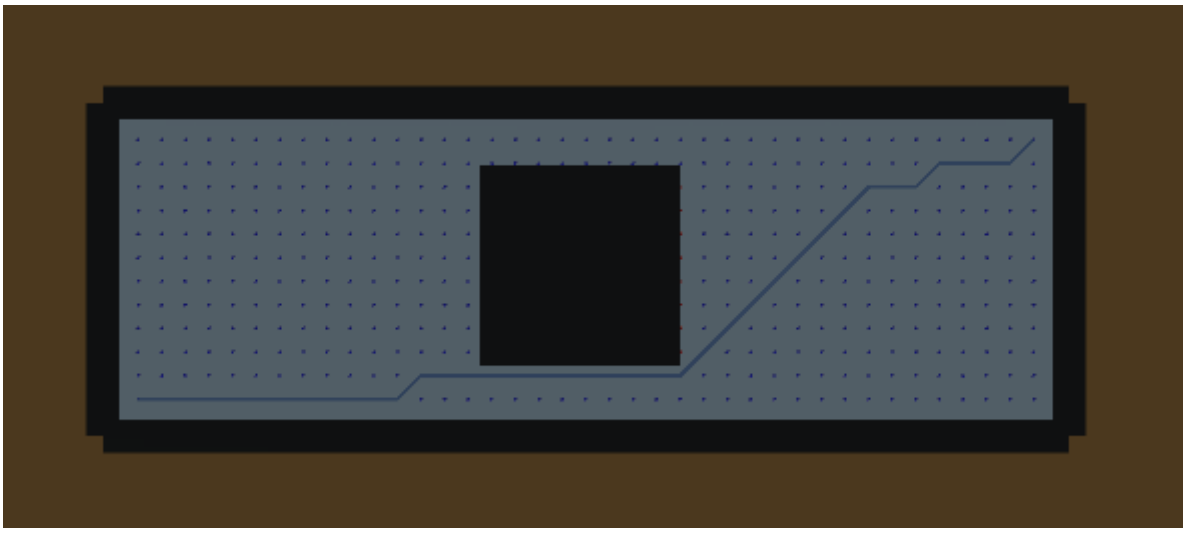

*Figure 15: Capture d'écran montrant la définition du chemin du joueur par l'algorithme A**

4.3 PLACEMENT DES GARDES SUR LA CARTE

Afin de placer au mieux les gardes à l'aide de cette méthode, il est possible de la diviser en plusieurs étapes. Ainsi, dans cette partie, nous allons voir la mécanique derrière le placement des gardes, leurs déplacements et la manière dont ils interagissent avec la matrice pour perturber le chemin.

4.3.1 DÉFINIR LA POSITION D'UN GARDE

Afin de définir la position d'un garde sur la carte, notre division de l'espace est centrée autour du chemin actuel du joueur. L'espace est alors divisé en deux grandes parties, la première autour du chemin défini par l'algorithme A* et la deuxième qui comprend tout le reste de la carte. Un point appartenant au chemin du joueur est choisi, compris dans les 70% au milieu du chemin. Cela a pour effet d'empêcher un garde d'apparaitre sur le point de départ ce qui rendrait le niveau impossible (cette condition est revérifiée plus tard et en tout temps afin d'éviter un point de départ impossible). Une fois le point choisi, une valeur aléatoire est ajoutée de telle manière à le décaler légèrement du chemin. Le point est ensuite adapté en coordonnées de matrices. Cette étape n'est pas obligatoire mais elle permet d'utiliser l'algorithme *A** pour définir le chemin des gardes si nécessaire. Cette étape nous permet donc de diviser l'espace en deux zones distinctes. La zone de placement se trouve autour du chemin du joueur, avec comme largeur cette valeur aléatoire ajoutée (Figure 16).

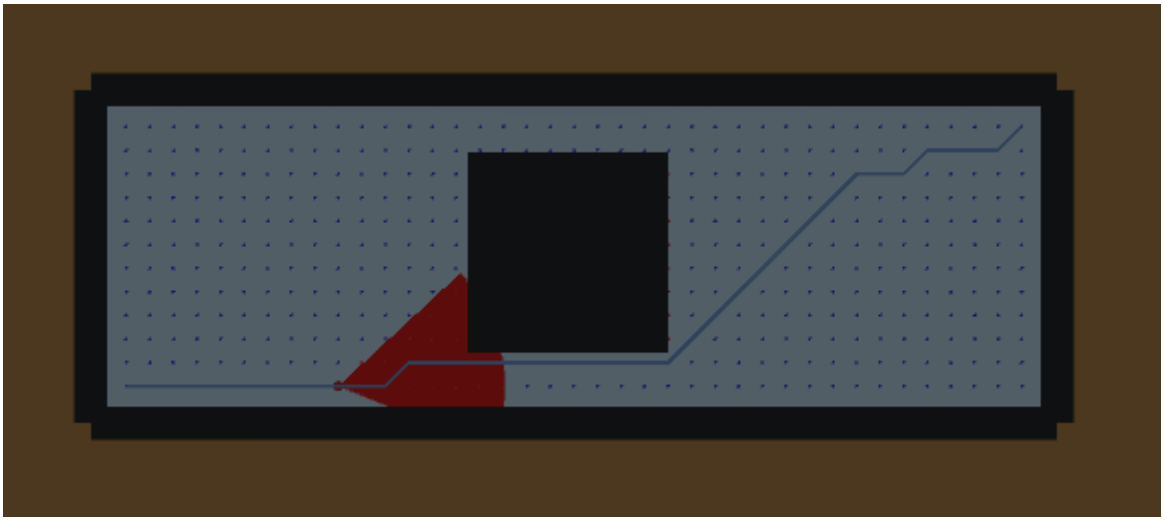

Figure 16: Capture d'écran montrant le placement initial du garde sur le chemin

4.3.2 DÉFINIR LE CHEMIN DES GARDES

Une fois le point initial choisi, il faut choisir dans quel sens le garde va se déplacer. Là encore, pour éviter les problèmes avec les points de départ et d'arrivée, le garde va se déplacer vers le centre du chemin. Un autre point est créé de la même manière que le premier, et après vérification de coordonnées et réajustement sur la matrice, un chemin est créé. Ce dernier est un simple segment s'il n'y pas d'obstacles entre les deux points, mais demande l'aide de l'algorithme A* dans le cas contraire. Il s'agit-là d'un point reprochable car les chemins donnés par l'algorithme A* suivent la matrice, ce qui donne en général des chemins très rigides. Le décalage aléatoire des points est tout le temps effectué mais comme les points sont remis en coordonnées de matrice après cette opération, il est possible et même assez courant d'avoir un chemin partant et arrivant sur des points du chemin (cf. Figure 17). Cette portion aléatoire est un facteur assez important sur la précision et l'utilité des gardes créés.

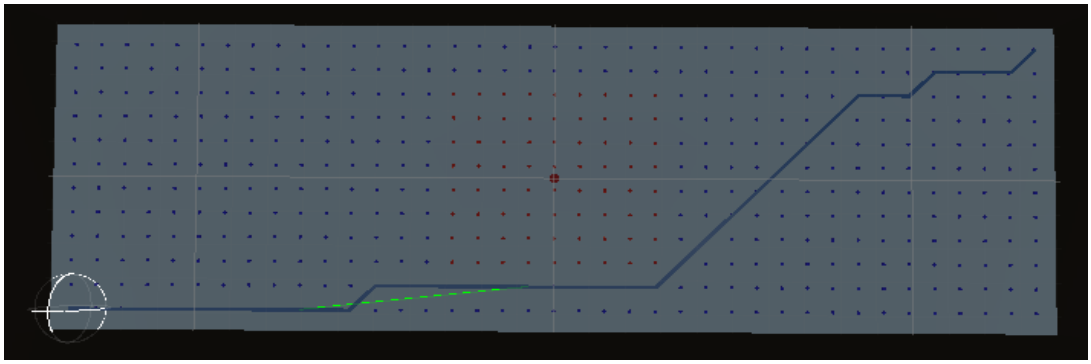

Figure 17: Capture d'écran de l'éditeur Unity montrant le chemin du garde en vert

4.3.3 PERMETTRE AUX GARDES DE BOUGER

Une fois les gardes placés, il faut leur donner la possibilité de se déplacer. Ils ont déjà en paramètre via leur création une vitesse de rotation et de déplacement qui sont tous les deux paramétrables. Un jeu d'infiltration plus complexe nécessitera sûrement plus de paramètres mais ces deux-là sont le minimum requis dans notre méthode de placement. Le chemin donné par les deux points définis jusque-là est transformé de manière à correspondre mieux à la structure matricielle et aux demandes de l'algorithme A*. Ainsi, le chemin sera transformé en calculant le temps que prend chaque action et envoyé vers l'IA du garde. Il est important de noter ici qu'un garde doit regarder le prochain point vers lequel il doit se déplacer avant d'effectuer l'action. On obtient donc une succession de rotation et de déplacement, le tout lié à un temps relatif par rapport à la longueur du chemin. Lorsque l'algorithme A* créé une nouvelle couche ou demande la mise à jour de l'une d'elle, il suffit alors de calculer la position et la rotation du garde pour le temps demandé. Les gardes vont alors mettre à jour le *mesh* ce qui va également changer l'état des points de la matrice.

4.4 BOUCLE DE JEU ET CALCUL DE LA DIFFICULTÉ

La dernière étape afin d'obtenir un algorithme complet est de créer une boucle dans laquelle l'algorithme va créer des gardes. Cette boucle va également apporter des modifications aux gardes car ces derniers suffisent rarement à couvrir une carte entière. Enfin un calcul de difficulté permet à l'algorithme de s'auto réguler et de s'arrêter lorsque le niveau atteint une valeur proche de celle demandée en entrée.

4.4.1 ANALYSE DES MODIFICATIONS DE CHEMIN

La première étape après avoir placé un garde est de trouver le nouveau chemin le plus court. Si l'algorithme A* ne donne pas de chemin, le niveau est considéré comme impossible et le garde est alors supprimé. En revanche, s'il existe un chemin, la différence de taille entre ce dernier et le précédent est enregistrée dans la valeur « d'importance » du garde.

Bien que l'étape soit simple en apparence, elle révèle un problème qui doit être résolu. On peut voir sur la Figure 18 le nouveau chemin calculé après le placement du garde. En le comparant avec celui de la Figure 16, on peut se rendre compte que les chemins font exactement la même taille. Ce cas peut se répéter sur beaucoup de cartes ayant une architecture semblable. D'autres cas liés notamment à la rotation des gardes au bout du chemin permettent également d'avoir des gardes ayant une importance de zéro. Dans ces deux cas, il peut être intéressant d'effectuer des manipulations supplémentaires pour essayer d'augmenter l'importance de ces gardes.

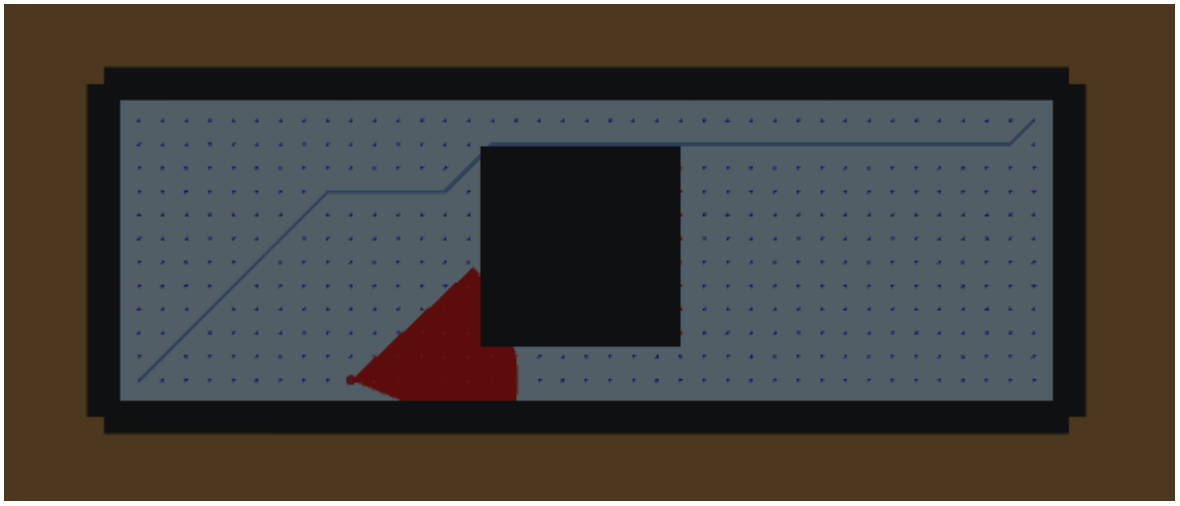

Figure 18: Capture d'écran vu de haut montrant la définition d'un nouveau chemin

4.4.2 BOUCLE DE CRÉATION DE GARDES

L'analyse des gardes possédants une importance de zéro est une tâche délicate et il existe deux situations qui peuvent mener à un tel résultat. La première, visible sur la Figure 18, est une carte pouvant être séparée en deux parties de distances égales avec un garde qui bloque le chemin le plus court d'un côté uniquement. Le deuxième cas serait que le décalage aléatoire appliqué sur les points résulte en un garde qui marche que d'un côté du chemin, ce qui, couplé à une rotation malchanceuse, résulterait en un évitement total du champ de vision du garde. Il serait possible d'essayer d'analyser chaque garde afin de conclure dans quel cas il se trouve mais il est plus simple d'appliquer la même procédure tout le temps et de déplacer ou non le garde selon le résultat. Cette procédure consiste à coupler les gardes afin de rendre leurs chemins plus complets. Cette méthode est d'ailleurs souvent utilisée dans les jeux d'infiltrations, permettant également d'augmenter la difficulté lorsque le joueur à la possibilité d'éliminer les gardes.

La première modification va ajouter un garde ayant un chemin parallèle et inversé à celui du garde d'origine. En prenant le chemin d'origine, en le modifiant de quelques unités

d’un côté et en inversant le point de départ et d’arrivée, on obtient un nouveau chemin qui permet au couple de garde de couvrir entièrement la zone autour du chemin d’origine.

Sur la Figure 19, le garde possède un placement utile mais limité. Il rallonge le chemin qui passe en dessous de l’obstacle au milieu, mais ne permet pas d’altérer celui d’en haut et obtient un score de zéro. Vient alors l’ajout du garde parallèle sur la Figure 20 qui va totalement changer la forme et la longueur du chemin le plus court, passant de 39 étapes à 45. Ce couple empêche totalement les chemins passant au-dessus de l’obstacle et rendent plus risqué le passage par en dessous, sans pour autant le rendre impossible. Comme il s’agit d’un succès, le même score d’efficacité est attribué aux deux gardes et l’algorithme peut alors continuer en plaçant un nouveau garde. Lorsque le placement parallèle n’est pas suffisant en revanche, l’algorithme va supprimer le grade créé et passer à la phase suivante.

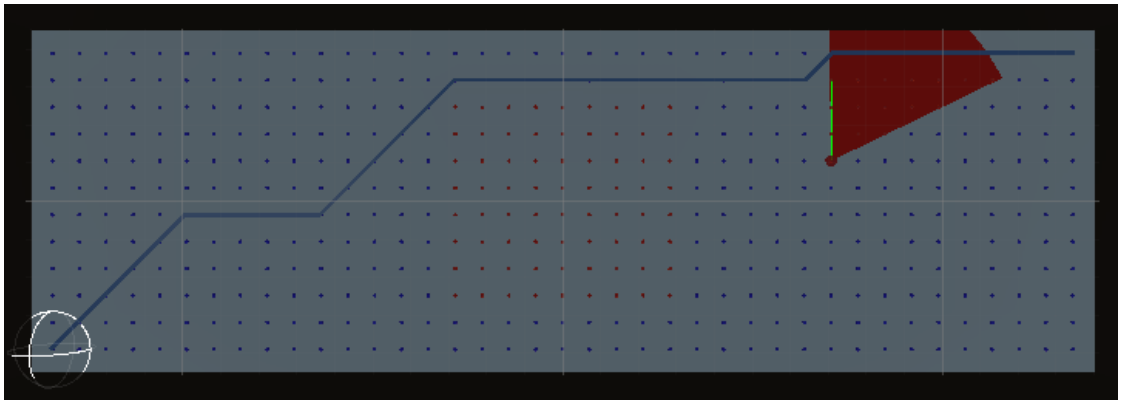

Figure 19 : Capture d’écran de l’éditeur Unity montrant le chemin du premier garde

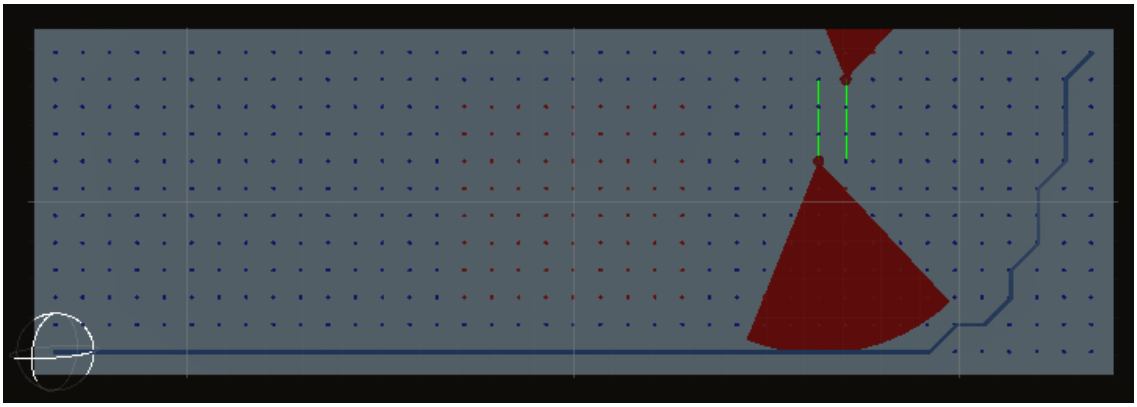

Figure 20 : Capture d'écran de l'éditeur Unity montrant le placement en parallèle du deuxième garde par rapport au premier

Cette deuxième modification couple le garde d'origine avec un autre ayant un chemin perpendiculaire au sien. Le deuxième chemin est construit pour intersecter le premier de manière perpendiculaire avec une taille similaire. Sur la Figure 21, le premier garde (en bas) a dévié le chemin, le faisant ainsi passer au-dessus de l'obstacle, néanmoins, l'ajout d'un garde supplémentaire en parallèle n'a pas changé le chemin.

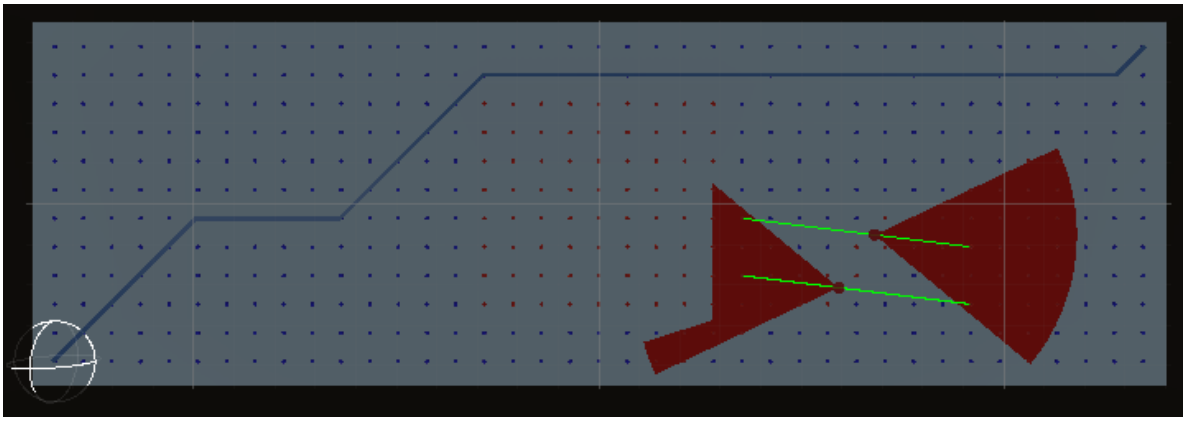

Figure 21 : Capture d'écran de l'éditeur Unity montrant un placement de garde parallèle possédant une efficacité de zéro

En ajoutant un garde perpendiculaire, on obtient un chemin légèrement plus long (cf. Figure 22). Le chemin ne parait pas plus long que celui de la Figure 21 mais il reste sur le même point quelques étapes cachées du champ du garde remontant derrière l'obstacle. De

plus, les deux cas présentés par la Figure 19 et la Figure 21 représentent les deux cas différents de garde ayant une valeur d'importance de 0 cités plus haut. Il est donc possible d'appliquer la même méthode et d'obtenir des résultats positifs.

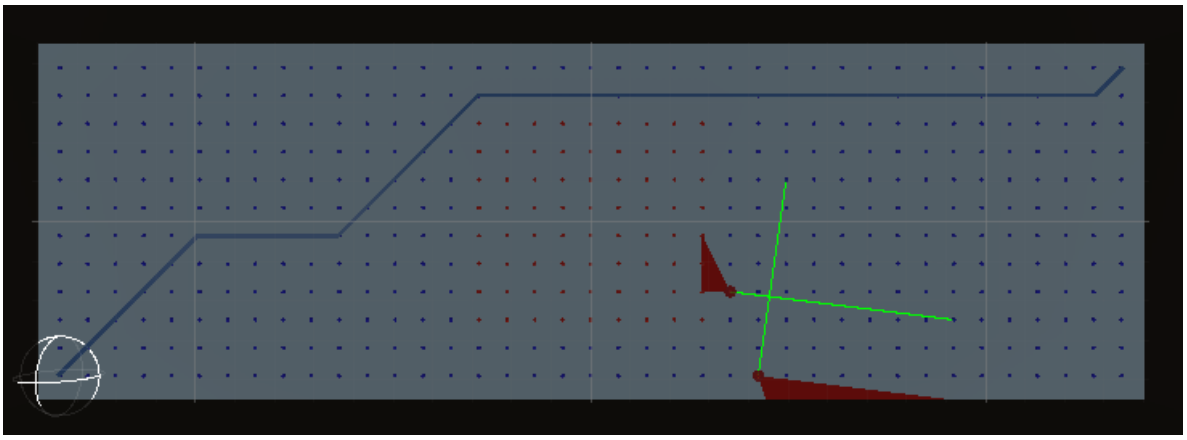

Figure 22 : Capture d'écran de l'éditeur Unity montrant le placement d'un garde avec un chemin perpendiculaire au premier

Finalement, lorsque même un garde perpendiculaire ne suffit pas à allonger le chemin, l'algorithme va replacer le garde ailleurs. Il serait possible de continuer à chercher différentes variantes mais garder l'algorithme simple permet aussi de mieux prédire son comportement.

4.4.3 CONDITIONS D'ARRÊT

La dernière étape afin d'obtenir une boucle finie est d'ajouter une condition d'arrêt. Comme mentionné précédemment, l'algorithme peut se baser sur une valeur de difficulté afin d'évaluer le placement des gardes. Il ne demande alors qu'un paramètre qui est la valeur de difficulté visée avant de lancer la boucle de construction. Ensuite, pour chaque garde placé, l'algorithme calcule la nouvelle difficulté du niveau et s'arrête lorsque la valeur est proche de celle demandée. La meilleure manière d'approcher ce paramètre ici était de s'inspirer de ce qui avait déjà été testé dans les jeux d'infiltrations (Tremblay, Torres, & Verbrugge, Measuring risk in stealth games., 2014). Un calcul de difficulté basé sur la

distance est simple à adapter à tout projet et c'est donc la première méthode qui va être implémentée. Nous voulons cerner le résultat final des formules entre 0 et 1 afin de mieux comparer les algorithmes de difficulté entre eux. Ainsi, il faut effectuer une opération mathématique pour que le résultat soit borné entre ces deux valeurs. Le calcul est fait en prenant la moyenne de distance entre tous les gardes et le joueur pour chaque étape du chemin le plus court actuel. Cette partie est assez simple à calculer et est obtenue via l'équation 1 avec l la longueur du chemin du joueur et g le nombre de garde.

$$mean = \frac{\sum_{t=0}^{l} \frac{\sum_{i=0}^{g} distance(posPlayer, posGarde_i)}{g}}{l}$$

1

Cette valeur représente déjà une difficulté. Plus elle est faible, plus la difficulté est grande car la distance moyenne entre le joueur et les gardes est plus petite. Néanmoins, afin d'obtenir une difficulté entre 0 et 1 avec une difficulté de 1 considéré comme la plus difficile, il faut appliquer une dernière opération avec une distance d qui correspond à la plus petite distance moyenne pour obtenir une difficulté de 1.

$$difficulty = \frac{d}{mean}$$

2

Il est encore possible d'obtenir des résultats supérieurs à 1 sur l'équation 2, mais ce problème peut être réglé en utilisant la fonction *clamp*, donnant un résultat borné entre 0 et 1. Il est également facile de la moduler car il n'y a qu'un seul paramètre d qui définit les limites de la fonction.

La structure du code permet également un autre moyen intéressant pour calculer la difficulté. Il est possible d'analyser les points de la matrice autour du joueur pour chaque étape du chemin. Dans un rayon de r points, il faut compter le nombre de points x qui sont

marqués comme « garde », c'est-à-dire dans le champ de vision d'un garde. Il est ensuite possible de calculer la moyenne pour le niveau entier ou même de trouver le pic de difficulté du niveau. Enfin, pour calculer la difficulté, il faut faire rentrer un dernier paramètre qui correspond au nombre minimum de points en moyenne autour du joueur afin d'obtenir une difficulté de 1. Ce paramètre p peut être obtenu via l'équation 3.

$$p = \frac{(r * 2 + 1)^2 - 1}{2}$$

3

Il s'agit-là d'un moyen parmi d'autres de le calculer mais la limite encore une fois est un paramètre qui peut être changé selon les besoins. Avec ce dernier paramètre il est ensuite assez simple de calculer la difficulté finale.

$$difficulty = \frac{mean}{p}$$

4

L'équation 4 est également simple à moduler car elle implique peu de paramètres et il est donc facile d'adapter la vitesse à laquelle la formule atteint la difficulté maximum.

Néanmoins, afin d'obtenir une difficulté exacte en tout temps, il manque un dernier élément. Comme l'algorithme va calculer la difficulté après avoir placé un garde, il se peut que le nouveau chemin ait été dévié pour passer de l'autre côté d'un obstacle par exemple, ce qui résulterait en une baisse plus ou moins importante de la difficulté, notamment lors de l'utilisation de la deuxième formule. En effet, cette dernière peut rapidement retomber vers une valeur de 0 à cause du manque de garde autour du joueur. Il faut donc essayer de compenser cette déviation en y ajoutant le paramètre d'importance que possède chaque garde. Pour ce paramètre, calculer la somme de l'importance des gardes et la diviser par la longueur du chemin actuelle permet d'obtenir une valeur utilisable que l'on peut juste ajouter

à la difficulté. En effet, même si la formule est elle-même bornée entre 0 et 1, il sera très rare qu'elle dépasse 0.5, ce qui signifierait que les gardes ont doublé la taille du chemin initial, montrant déjà que le niveau s'annonce difficile. S'il ne change pas beaucoup le résultat de chemins complexes, ce paramètre permet en revanche de renforcer la difficulté des gardes ayant modifié le chemin de manière drastique tout en éloignant le joueur de ces derniers. Ce paramètre n'est en revanche pas obligatoire pour la première formule car la moyenne de distance ne pourra jamais être nulle.

A l'aide de ces formules, l'algorithme peut s'arrêter tout en donnant un résultat proche de ce qui est demandé. Afin de mieux visualiser cette boucle, la Figure 23 représente son pseudo code. On peut y voir notamment la gestion des *Pair* qui correspond aux couples de gardes parallèles et perpendiculaires. Il y a également une autre condition d'arrêt au début symbolisé par la variable « *triesNewGuard »* qui correspond au nombre de nouveau garde placé sans améliorations. Il s'agit-là d'une sécurité supplémentaire pour éviter les boucles infinies.

Algorithm 1 Guard Placement

```
1: procedure NewPathPlacement
2:     if triesNewGuard > 10 then return
3:     end if
4:     if Abs(levelDifficulty − aimedDifficulty) < 0.05 then return
5:     end if
6:     if levelDifficulty − aimedDifficulty > 0.15 then
7:         CancelGuard(lastGuardID)
8:     end if
9:     if Path has been updated then                ▷ False on the initial path
10:        triesNewGuard ← 0
11:        if Pair! = null then
12:            Pair.First.Importance ← Pair.Second.Importance
13:            if Pair.First.Importance = 0 then
14:                CancelGuard(Pair.Second)
15:            end if
16:            Pair ← null
17:        end if
18:    end if
19:    if No Guard then
20:        PlaceNewGuard()
21:    end if
22:    guard ← listOfGuards[0]
23:    if guard.Importance = 0 then
24:        if guard.Changes = 0 then
25:            PlaceNewGuard(Parallel, guard.ID)
26:        else if guard.Changes = 1 then
27:            PlaceNewGuard(Cross, guard.ID)
28:        else if guard.Changes >= 5 then
29:            CancelGuard(guard.ID)
30:        else
31:            ChangeGuardPath(guard.ID)
32:        end ifreturn
33:    end if
34:    PlaceNewGuard()
35: end procedure
```

Figure 23: Pseudocode de la fonction de la boucle principale de placement des gardes

CHAPITRE V – ÉVALUATION DE L'APPROCHE

Dans ce chapitre, nous allons étudier les différents tests et résultats permettant d'évaluer les performances des algorithmes. Pour cela, nous allons tout d'abord voir les niveaux utilisés pour les tests puis une explication de ces derniers avant d'en analyser les résultats.

5.1 NIVEAUX DE TEST

La génération des cartes est contrôlée par une graine permettant de répéter les expériences si besoin. Ici, plus de cinq cents configurations ont été analysés et seuls six ont été retenues pour les tests. Ces configurations possèdent des obstacles qui influencent le chemin du joueur et qui n'ont pas d'obstacles trop excentrés. En plus de la carte représentée sur la Figure 13 (nommé « 29 » en référence à la graine permettant de la générer), cinq autres cartes « 75 », « 216 », « 331 », « 557 » et « 561 » seront utilisées pour les tests.

Les deux cartes sur la Figure 24 sont bien différentes de la carte 29, avec deux obstacles au lieu d'un et un chemin plus sinueux et plus dépendant de ces obstacles. Il est aussi plus compliqué d'envisager plusieurs chemins de même taille passant à différents endroits de la carte. Ces cartes sont donc possiblement plus intéressantes car plus proches de la réalité des jeux d'infiltrations. Les trois autres cartes représentent en quelque sorte des versions plus grandes et plus complexes des trois déjà existantes.

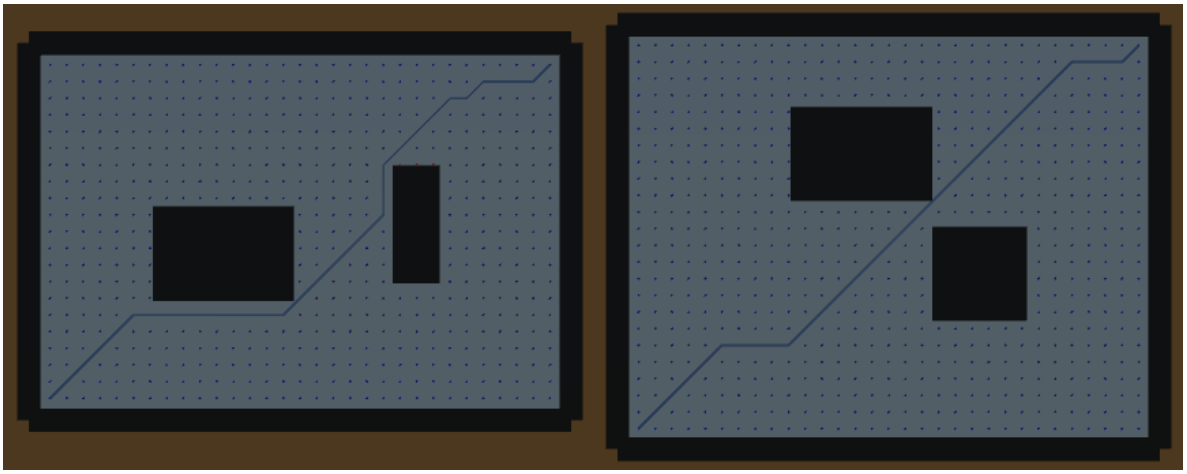

Figure 24: Capture d'écran montrant le chemin optimal d'origine pour les cartes 75 (à gauche) et 216 (à droite)

La carte « 331 » présente sur la Figure 25 offre trois chemins de même taille au joueur. En effet, si l'algorithme donne ce chemin particulier, il existe deux autres chemins de même taille passant en haut et au milieu des deux obstacles.

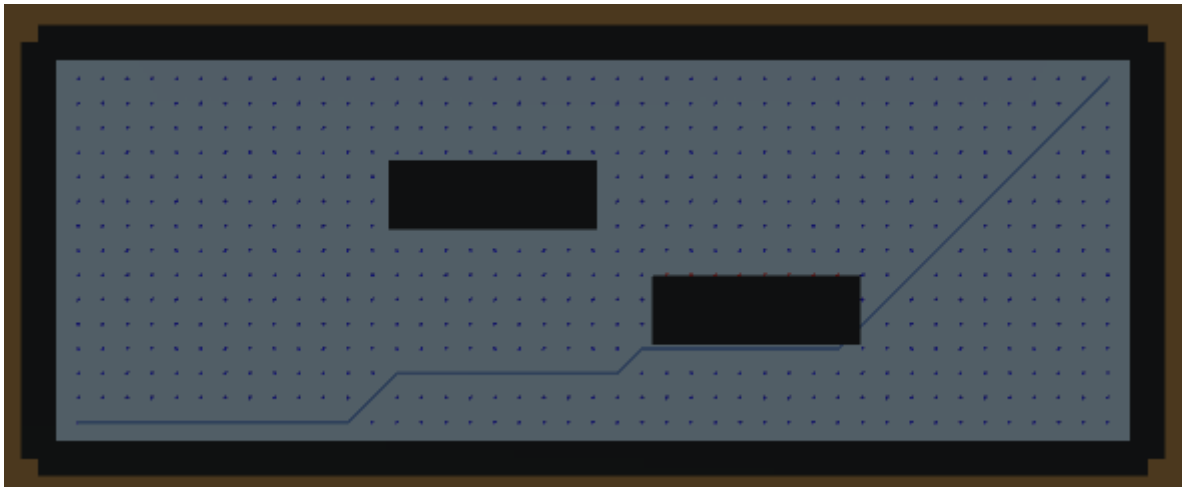

Figure 25 : Capture d'écran montrant le chemin optimal d'origine sur la carte 331

Enfin les cartes « 557 » et « 561 » présentent sur la Figure 26 offrent des grands espaces et pourront donc accueillir une plus grande quantité de gardes.

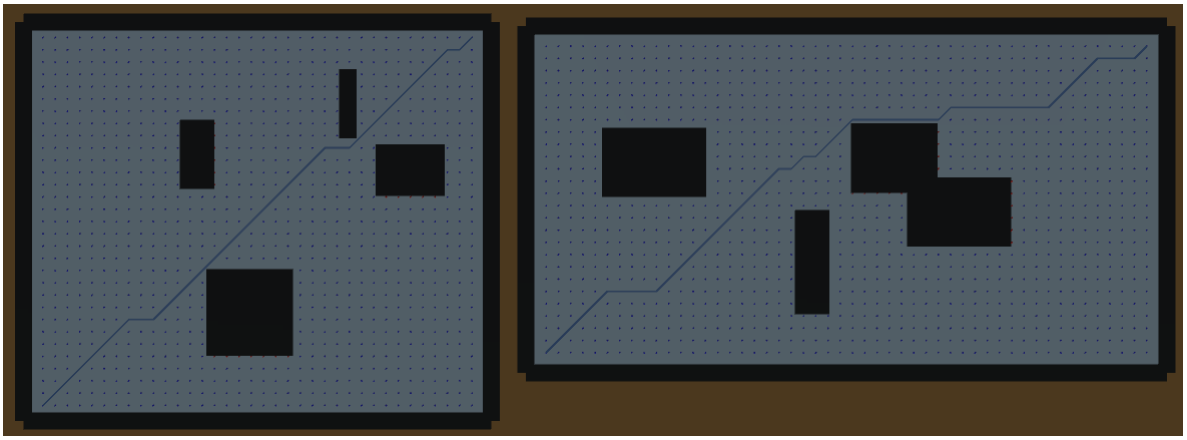

Figure 26 : Capture d'écran montrant le chemin optimal d'origine sur la carte 557 (à gauche) et 561 (à droite)

5.2 TESTS SUR LE PLACEMENT

Le premier test a pour but d'observer l'efficacité du placement des gardes ainsi que les deux étapes « parallèle » et « perpendiculaire » qui sont liées. Pour une trentaine de générations, on observe à partir de quelle étape le chemin d'origine est allongé. Si la formation perpendiculaire n'augmente pas la taille du chemin, on accepte qu'aucun garde n'a modifié le chemin (cela correspond virtuellement à un nouveau garde) et on recommence l'expérience.

Tableau 2: Etapes de l'algorithme allongeant pour la première fois le chemin initial du joueur

Identifiant de la carte	Garde seul	Parallèle	Perpendiculaire	Aucun	Total
29	4	12	4	10	30
75	16	2	6	6	30
216	25	5	0	0	30
331	6	4	0	20	30
552	26	2	0	2	30
561	10	8	6	6	30

Le Tableau 2 montre les résultats de ce test. La variation entre les résultats pour chaque carte nous indique déjà qu'il est compliqué d'évaluer la performance du placement car celle-ci dépend entièrement de la carte. Il est en effet assez simple d'obtenir des gardes

qui n’arrivent pas à faire dévier le chemin sur la carte 29 ou la 331 car leurs architectures permettent non seulement une multitude de chemins de même taille mais également l’impossibilité pour certains placements d’influencer ces chemins. Néanmoins, dans le cas de la carte 29, il est intéressant de noter que l’ajout d’un garde parallèle permet de modifier une quantité non négligeable de chemins. Ce résultat montre que la méthode de placement parallèle peut être une bonne solution pour couvrir des grands couloirs. Les deux cartes 75 et 561 possèdent de loin les résultats les plus équilibrés. De manière générale, ces cartes n’ont qu’un seul chemin de taille minimum mais la manière dont est codé le placement de gardes permet de temps en temps à un garde de se créer de manière assez excentrée par rapport au chemin à cause d’un obstacle. La formation parallèle ou perpendiculaire n’impacte alors pas forcément le chemin car gênée par cet obstacle. La carte 75 montre également que le placement perpendiculaire semble être légèrement plus efficace que le parallèle sur cette carte reflétant de son utilité lors de ces chemins excentrés. Enfin les cartes 216 et 552 nous informe que le placement initial des gardes marche très bien sur les passages étroits. A noter également que le chemin, même modifié par le garde passait encore généralement au même endroit, en attendant seulement que le garde se tourne pour continuer. C'est dans ces conditions qu’il pourrait être intéressant d’ajouter un garde parallèle même si le précédent a modifié le chemin.

5.3 TESTS SUR LA DIFFICULTÉ

La deuxième phase de test concerne les formules de difficultés. Comme chaque formule donne une difficulté entre 0 et 1, il est possible de comparer leurs évolutions. Il est en revanche plus compliqué de dire si une méthode fonctionne mieux qu’une autre car les valeurs 0 et 1 ne correspondent pas à un placement particulier. Chaque formule va obtenir des valeurs différentes et l’algorithme va donc placer plus ou moins de gardes selon la

vitesse à laquelle la valeur demandée est atteinte. On peut néanmoins faire le graphique de ces deux fonctions afin de comparer la possible évolution de chacune.

Afin d'obtenir les courbes sur la Figure 27, la distance d pour la méthode de distance est 6, résultant donc en une difficulté globale de 1 lorsque la distance moyenne est de 6. Pour l'autre méthode, la distance r est de 2, l'analyse se fait donc sur un carré de 5 par 5 autour du joueur et le nombre afin d'obtenir une difficulté de 1 p est 12. La courbe est représentée sans l'ajout de l'importance des gardes car elle ne change pas l'allure de la courbe et est compliquée à prédire sur ce graphique.

Tout d'abord, le point important à remarquer est qu'elles n'évoluent pas dans le même sens. La difficulté de la première méthode va diminuer lorsque que la distance va augmenter alors que celle de la deuxième va augmenter avec le nombre de points marqués autour du joueur. Cela nous amène également à observer que la deuxième courbe peut atteindre une difficulté de 0. C'est la raison pour laquelle il faut ajouter l'importance des gardes dans la formule. On peut néanmoins le voir comme un désavantage pour la première méthode qui peut subir des effets similaires, en éloignant le chemin du garde placé, et ainsi diminuer drastiquement la difficulté d'une étape à l'autre. Il est possible selon les résultats d'ajouter ce paramètre de manière pondérée à cette formule de distance.

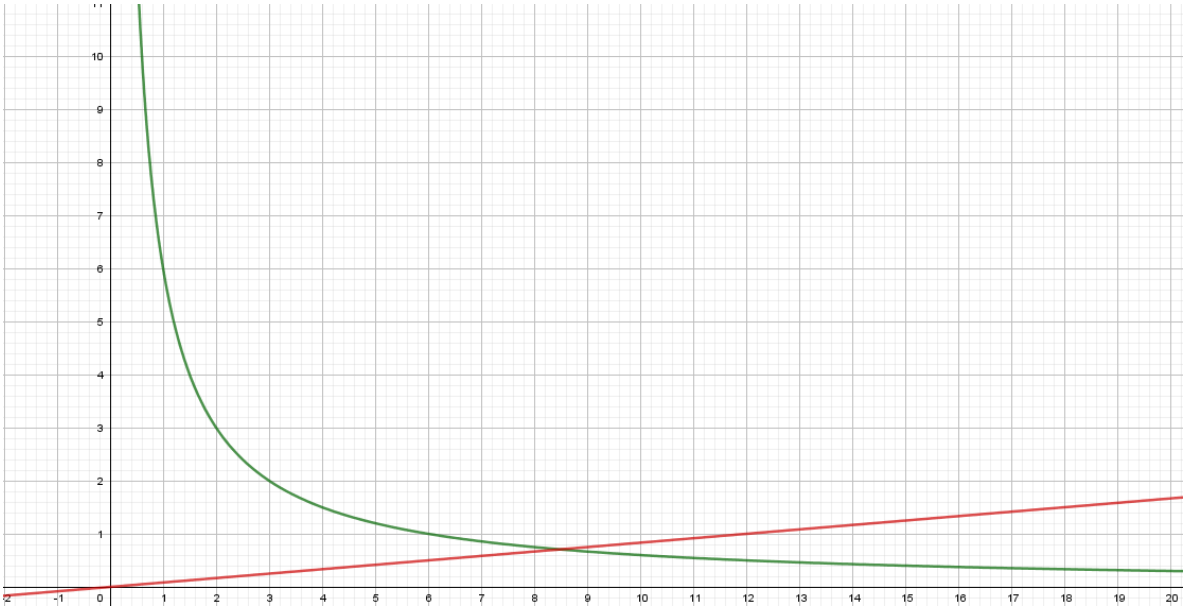

Figure 27: Capture d'écran du logiciel GeoGebra montrant l'allure des courbes des fonctions de difficulté avec la méthode de distance en vert et la méthode de points en rouge

Le test est assez simple et direct. En fixant la difficulté voulue à 1, on laisse l'algorithme construire les gardes et pour chaque garde confirmé (dont l'importance n'est pas 0), on note la valeur de difficulté ainsi que la configuration du ou des gardes qui viennent d'être placé. La Figure 28 montre les résultats de ce test sur la carte 29. On peut noter en plus des deux courbes de difficulté la présence des courbes « importance » et « points – importance ». Ces courbes représentent la valeur brute de la méthode de point ainsi que celle de la moyenne d'importance des gardes. L'addition de ces deux courbes permet donc d'obtenir celle des points, et la séparer permet de voir l'importance de chaque. On peut observer que même si la méthode de distance commence bien plus haut, elle diminue légèrement en ajoutant des gardes. En fait la formule possède un inconvénient assez gros.

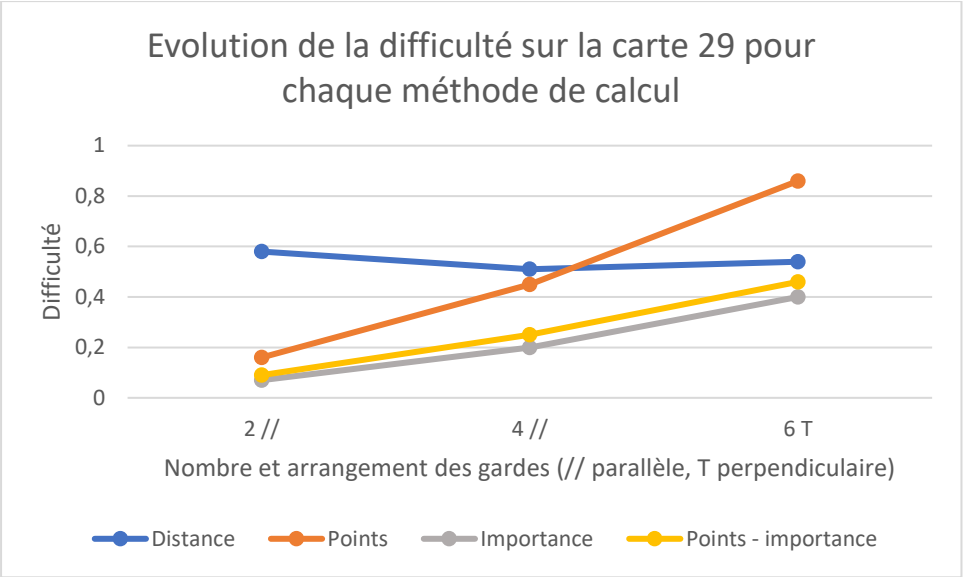

Figure 28: Graphique montrant l'évolution de la difficulté pour chaque méthode sur la carte 29

En comptant tous les gardes dans le calcul de distance, le résultat est influencé négativement par tous les gardes loin du joueur, ce qui ne peut pas être rattrapé par ceux qui sont proches. Afin d'obtenir un résultat croissant il faudrait prendre en compte seulement le ou les gardes les plus proches du joueur en tout temps. L'approche par point semble en revanche déjà plus adaptée. La difficulté augmente peut-être même trop vite avec les gardes du fait que l'importance augmente en parallèle. Il est possible là aussi d'envisager des changements notamment en pondérant la valeur de l'importance des gardes de manière à moins influencer le résultat final. Il faut néanmoins surveiller ce paramètre car l'algorithme s'est arrêté sur une difficulté de 0.86 et cela correspond plus ou moins à la limite de gardes que la carte acceptait avant de bloquer tous les chemins possibles.

Grace à cette analyse, il est possible d'effectuer un nouveau test légèrement modifié. Le premier point à changer est la formule sur la méthode de distance, en ne prenant que le garde le plus proche du joueur pour chaque étape au lieu de prendre la moyenne. Un changement de carte est également intéressant car la carte 29 ne laisse pas forcément assez de gardes pour bien observer la progression de la courbe.

La Figure 29 illustre les résultats de ce deuxième test et l'on peut voir au nombre de gardes que le changement de carte s'est avéré utile. La carte « 75 » est plus spacieuse et offre plus de possibilités au joueur pour contourner les gardes. De ce fait, l'algorithme a besoin de plus de gardes pour bloquer le chemin du joueur ou atteindre une difficulté de 1. On peut également observer que la nouvelle méthode de calcul par distance dépasse la limite de difficulté de 1. C'est volontaire afin d'observer sa croissance même en dehors de ces valeurs. Cette nouvelle courbe de distance ne prenant en compte que le garde le plus proche suit une évolution similaire à la courbe de points à part pour le dernier placement de garde. Les étapes où la difficulté diminue correspondent à un changement de chemin d'un coté à l'autre de la carte et la formule des points y est moins impactée car elle prend en compte l'importance des gardes. Cette valeur ne fait qu'augmenter avec le temps et permet d'accompagner positivement le calcul des points. Ici, elle a plus d'influence que pour la carte 29, elle-même étant plus grande, les chemins possibles du joueur sont ainsi plus éloignés. La difficulté est alors plus susceptible de baisser d'une étape à l'autre alors que la valeur d'importance reste croissante.

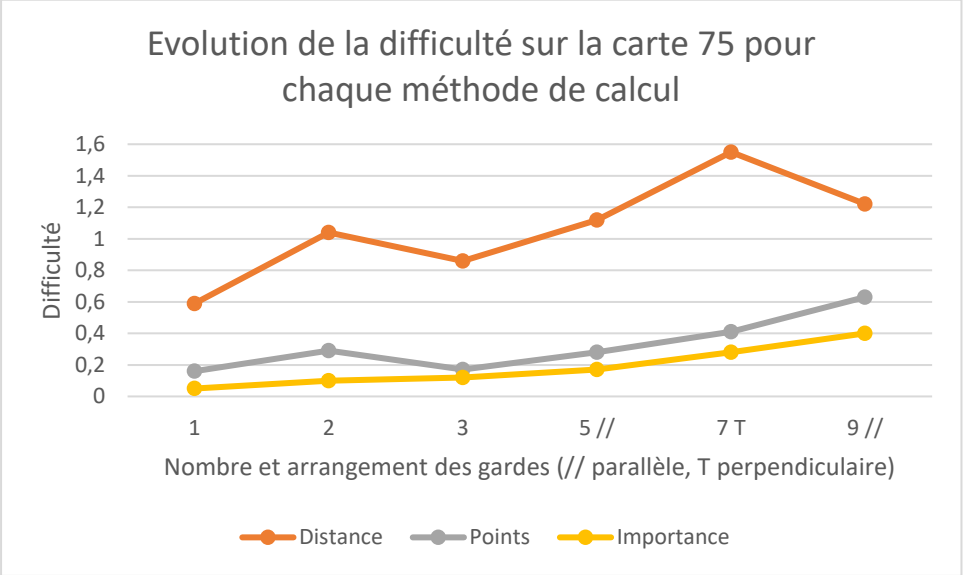

Figure 29 : Graphique montrant l'évolution de la difficulté pour chaque méthode sur la carte 75

Ce deuxième test nous permet de voir que la première formule de distance était une approche assez naïve du problème, et que ses défauts ont été comblés par la nouvelle formule. Cela nous amène également à effectuer les tests sur les autres cartes afin d'en analyser les résultats. La carte 331 n'est pas utilisée pour ces tests, car le Tableau 2 montre qu'il s'agit d'une carte où il est compliqué de bien placer des gardes et de bons résultats seront compliqués à obtenir.

La Figure 30, représentant l'évolution des difficultés sur la carte 216 montre plusieurs points intéressants. Le premier est ce pic obtenu lors du placement du deuxième et troisième garde en parallèle. La structure en matrice autorise certaines imperfections car il n'y a aucune vérification pour savoir si le mouvement entre deux couches est possible. L'algorithme ne fait que vérifier si le point d'arrivée de chaque déplacement est libre et cela peut mener à des chemins passants à travers un garde tournant trop vite par exemple. C'est ici le cas, et la moyenne de distance a donc drastiquement diminué résultant en un pic de difficulté. L'étape d'après montre une valeur bien plus faible, signe d'un changement de chemin. Le deuxième point intéressant est la chute de difficulté de la méthode distance après l'étape 2. Là encore on peut observer l'efficacité de l'ajout de l'importance aux gardes, qui compose 90% de la valeur totale de la difficulté sur l'étape 4 (6 gardes) avant de se stabiliser et de laisser la méthode reprendre une partie de ce résultat.

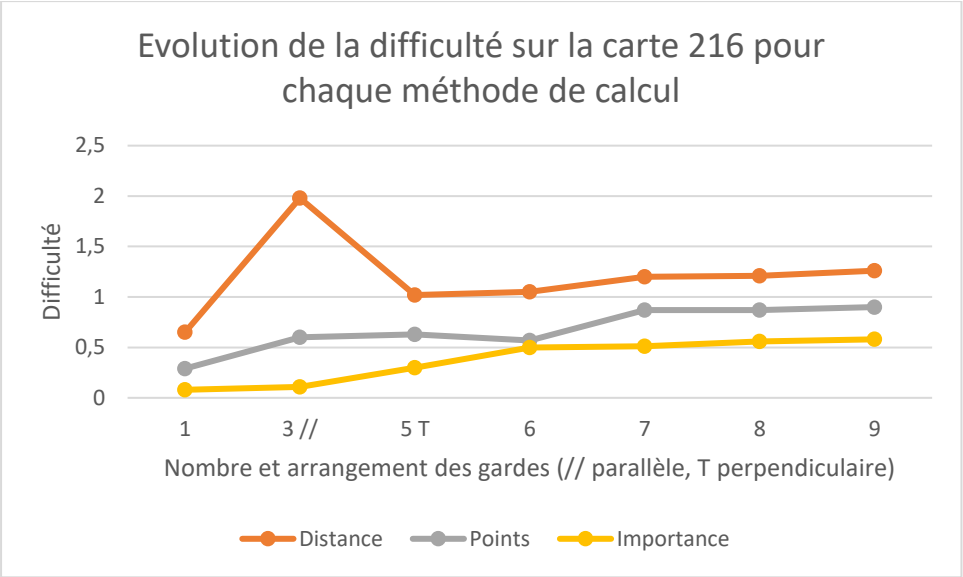

Figure 30 : Graphique montrant l'évolution de la difficulté pour chaque méthode sur la carte 216

Les deux derniers tests sur les cartes 557 et 561, représentés sur les Figure 31 et Figure 32 ne possèdent pas d'extravagances, et les observations pouvant en être tirés viennent rejoindre celles faites sur la carte 75 (Figure 29).

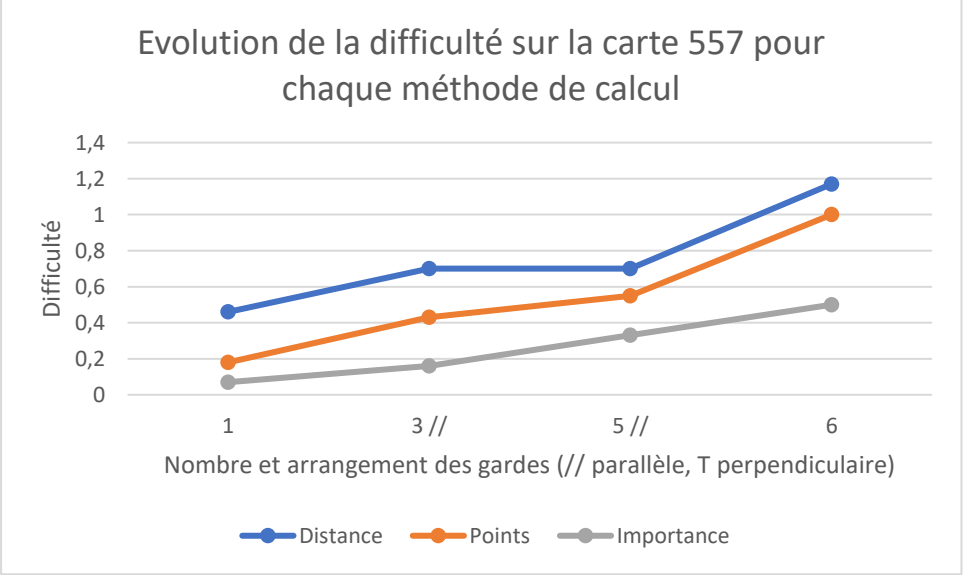

Figure 31 : Graphique montrant l'évolution de la difficulté pour chaque méthode sur la carte 557

Ces 4 derniers graphiques montrent que cette nouvelle formule de distance dépasse assez souvent la valeur de 1 avant la formule de points. Néanmoins les deux formules évoluent de la même manière, chacune possédant des particularités et il est impossible à ce stade de dire si l'une est meilleure que l'autre. Des tests comparant le comportement humain aux deux formules pourraient permettre de faire un choix mais il est également possible que chaque formule donne des résultats plus ou moins bons selon la carte.

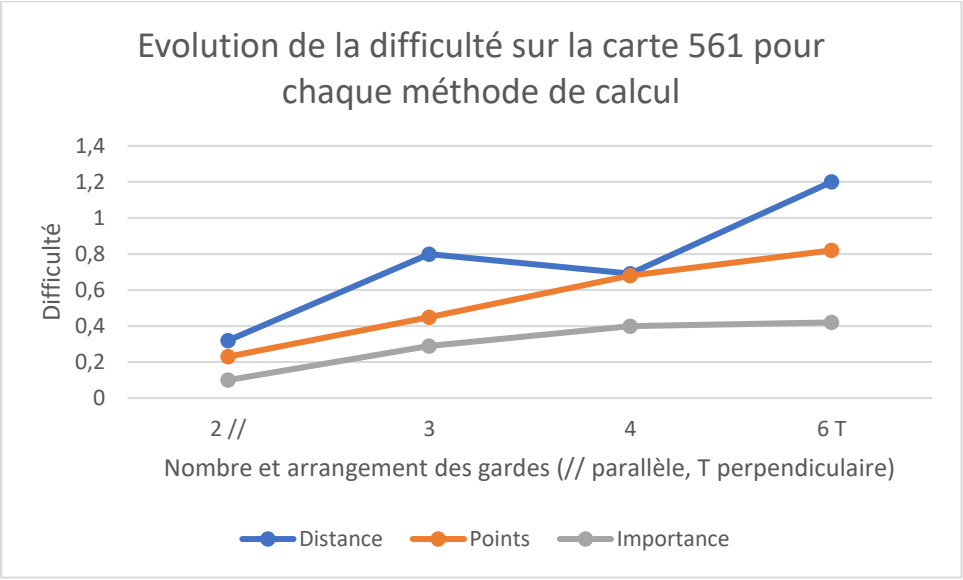

Figure 32 : Graphique montrant l'évolution de la difficulté pour chaque méthode sur la carte 561

CONCLUSION

Le placement et la définition de chemins représentent le cœur d'une phase d'infiltration. L'évolution des jeux fait qu'il est maintenant possible d'intégrer une grande quantité de mécaniques dans un seul jeu et de laisser le joueur choisir son style de jeu. Cela peut amener bien des défis du côté des concepteurs qui doivent penser aux nombreuses possibilités qui s'ouvrent. C'est particulièrement le cas avec la mécanique d'infiltration qui nécessite une grande attention lors de la conception. Les niveaux doivent être réfléchis aussi bien dans leurs architectures, leurs gardes et les gadgets disponibles. Tout cela demandant une expérience particulière qu'il n'est pas simple d'acquérir.

Notre algorithme s'est montré assez performant dans le placement des gardes. La division de l'espace en s'aidant du chemin du joueurs ne correspond peut-être pas à toutes les situations mais permet de disperser les gardes à travers la carte lorsque celle-ci offre un espace assez grand. Le placement de gardes unique s'est avéré efficace sur une configuration adaptée. Comme les gardes modifient le chemin à chaque placement, nous obtenons une division de l'espace dynamique. Le chemin évite les gardes et le placement des gardes se fait donc loin de ceux déjà placés. Les placements parallèles ainsi que perpendiculaires quant à eux ont obtenus des bons résultats mais nécessitent une étude plus approfondie. Les gardes parallèles permettent de combler l'aléatoire dans la définition du chemin des gardes. La formation perpendiculaire quant à elle permet de mieux gérer les défauts dans l'aléatoire concernant le placement des gardes. Finalement, la gestion totale de l'algorithme régulé par la difficulté est un succès également. Cette méthode empêche des niveaux trop simples ou impossibles et permet un placement adapté à la demande de difficulté initiale.

Il reste néanmoins des points sur lesquels il est possible de travailler. Les gardes possédant une faible importance peuvent être mieux exploités. Analyser chaque garde pourrait être une opération couteuse mais utile afin de voir comment est modifié le chemin. Le placement même des gardes ainsi que la définition de leurs chemins et de leurs comportements de manière générale peuvent également être améliorés. Le placement parallèle et perpendiculaire aide à combler les défauts de l'aléatoire mais cette approche est encore assez triviale et pourrait être améliorée. L'utilisation d'une intelligence artificielle n'est pas à exclure et pourrait aider à produire des gardes plus réalistes. Pour le moment, les chemins sont définis par des lignes droites, et les rotations des gardes se font toujours dans le même sens en suivant le comportement des fonctions mathématiques de Unity (notamment l'interpolation linéaire entre deux angles), ce qui limite la diversité dans le comportement des gardes. Enfin, le calcul de difficulté, bien qu'efficace commence un peu haut et peut encore être amélioré pour laisser plus de liberté sur le résultat, ce qui résulterait en un comportement plus prévisible.

BIBLIOGRAPHIE

Al Enezi, W., & Verbrugge, C. (2023). Investigating the influence of behaviors and dialogs on player enjoyment in stealth games. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, (pp. 166-174).

Alix, A. (2005). Beyond P-1: Who Plays Online?. *DiGRA Conference.*

Attali, D. (1995). Squelettes et graphes de Voronoi 2D et 3D. Université Joseph-Fourier-Grenoble I.

Boocock, S. S., & Schild, E. O. (1968). *Simulation games in learning.* ERIC.

Coburn, J. Q., Freeman, I., & Salmon, J. L. (2017). A Review of the Capabilities of Current Low-Cost Virtual Reality Technology and Its Potential to Enhance the Design Process. *Journal of Computing and Information Science in Engineering*.

Collins, R. T., Lipton, A. J., Kanade, T., Fujiyoshi, H., Duggins, D., Tsin, Y., . . . Burt, P. (2000). A system for video surveillance and monitoring. *VSAM final report*, 1-68.

Delling, D., Sanders, P., Schultes, D., & Wagner, D. (2009). Dans a. a. Algorithmics of large and complex networks: design, *Engineering route planning algorithms* (pp. 117-139). Springer.

Dillon, T. (2005). Adventure games for learning and storytelling. *UK, Futurelab Prototype Context Paper, Adventure Author*.

Djaouti, D. (2019). *Préhistoire du jeu vidéo.* LudoScience.

Haering, N., Venetianer, P. L., & Lipton, A. (2008). The evolution of video surveillance: an overview. *Machine Vision and Applications*, 279-290.

Hoffmann, F., Kaufmann, M., & Kriegel, K. (1991). The art gallery theorem for polygons with holes. *Proceedings 32nd Annual Symposium of Foundations of Computer Science*, (pp. 39-48). San Juan.

Ko, T. (2008). A survey on behavior analysis in video surveillance for homeland security applications. *2008 37th IEEE Applied Imagery Pattern Recognition Workshop*, (pp. 1-8).

Koster, R. (2013). *Theory of fun for game design.* O'Reilly Media, Inc.

Mirza-Babaei, P., Moosajee, N., & Drenikow, B. (2016). Playtesting for Indie Studios. *Proceedings of the 20th International Academic Mindtrek Conference*, (pp. 366-374).

Pallavicini, F., Pepe, A., & Minissi, M. E. (2019). Gaming in Virtual Reality: What Changes in Terms of Usability, Emotional Response and Sense of Presence Compared to Non-Immersive Video Games? *Simulation & Gaming*, 136-159.

Russell, S. J. (2010). *Artificial intelligence a modern approach.* Pearson Education, Inc.

Singh, D. (2015). *Using medial skeleton for path finding in dynamic stealth games.* Montréal: McGill University (Canada).

Smith, R. (2006). Level building for stealth gameplay. *Ronin Game Developer*.

Stamenković, D., Jaćević, M., & Wildfeuer, J. (2017). The persuasive aims of Metal Gear Solid: A discourse theoretical approach to the study of argumentation in video games. *Discourse, Context & Media*, 11-23.

Sweetser, P., Johnson, D., Wyeth, P., Anwar, A., Meng, Y., & Ozdowska, A. (2017). GameFlow in different game genres and platforms. *Computers in Entertainment (CIE)*, 1-24.

Tremblay, J., Torres, P. A., & Verbrugge, C. (2014). An algorithmic approach to analyzing combat and stealth games. *2014 IEEE Conference on Computational Intelligence and Games*, (pp. 1-8).

Tremblay, J., Torres, P. A., & Verbrugge, C. (2014). Measuring risk in stealth games. *Foundations of Digital Games.*

Tremblay, J., Torres, P. A., Rikovitch, N., & Verbrugge, C. (2013). An exploration tool for predicting stealthy behaviour. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, (pp. 34-40). Boston.

Wood, K. R. (2011). *Simulation video games as learning tools: An examination of instructor guided reflection on cognitive outcomes.* Georgia State University.

Xu, Q., Tremblay, J., & Verbrugge, C. (2014). Generative methods for guard and camera placement in stealth games. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, (pp. 87-93). Raleigh.

Xu, Q., Tremblay, J., & Verbrugge, C. (2014). Procedural guard placement for stealth games. *Proc. of the 5th workshop on Procedural Content Generation (PCG).*

Zagal, J. P., & Mateas, M. (2010). Time in Video Games: A Survey and Analysis. *Simulation & Gaming*, 844-868.

Zeng, W., & Church, R. L. (2009). Finding shortest paths on real road networks: the case for A. *International journal of geographical information science*, 531-543.

ANNEXE 1

Fonction permettant la génération d'un chemin de garde

```
List<Vector3> GenerateNewPath()
{

    int indexPoint = Random.Range((int)(pathLength * 0.15f), (int)(pathLength * 0.85f));

    Vector3 firstPoint = currentPath[indexPoint] + Random.insideUnitSphere * Random.Range(1, 2f);
    firstPoint.y = 0;
    firstPoint = matrixManager.GetClosestFreePoint(firstPoint, guardPlacementConditions, saveGuardRange);

    //Between 7% and 12% of the total path
    float rangeIndex = Random.Range(pathLength * 0.15f, pathLength * 0.21f);
    float rangeInGame = rangeIndex * matrixManager.matrixInDistance;

    int indexDirection;
    if (indexPoint <= pathLength / 2)
    {
        indexDirection = indexPoint + (int)rangeIndex;
    }
    else
    {
        indexDirection = indexPoint - (int)rangeIndex;
    }

    //Take a point with range calculated in direction on the indexDirection
    Vector3 secondPoint = new Vector3();
    do
    {
        secondPoint = currentPath[indexPoint] + Random.insideUnitSphere * Random.Range(1, 1.2f) +
        (currentPath[indexPoint] - currentPath[indexDirection]).normalized * rangeInGame;

        secondPoint.y = 0;
        secondPoint = matrixManager.GetClosestFreePoint(secondPoint, guardPlacementConditions, saveGuardRange);
    //To avoid two points being too close
    } while (secondPoint == firstPoint || Vector3.Distance(firstPoint, secondPoint) < 1.5f);

    List<Vector3> path = CorrectPath(firstPoint, secondPoint);

    for (int i = 0; i < path.Count - 1; i++) //Debug
    {
        Debug.DrawLine(path[i], path[i + 1], Color.green, 100f);
    }

    return path;
}
```

ANNEXE 2

Fonction permettant de créer un chemin parallèle

```
List<Vector3> ParallelWithOtherGuard(GuardTracker guard)
{
    Vector3 otherFirstDirection = guard.Path[1] - guard.Path[0];
    int randomFactor = Random.Range(0, 2) < 1 ? -1 : 1;
    Vector3 directionVector =
        new Vector3(otherFirstDirection.z * randomFactor, 0, -otherFirstDirection.x * randomFactor).normalized;

    directionVector *= Random.Range(1, 3);

    Vector3 firstPoint = guard.Path[0] + directionVector;
    Vector3 secondPoint = guard.Path[guard.Path.Count - 1] + directionVector;

    firstPoint = matrixManager.GetClosestFreePoint(firstPoint, guardPlacementConditions, saveGuardRange);
    secondPoint = matrixManager.GetClosestFreePoint(secondPoint, guardPlacementConditions, saveGuardRange);

    List<Vector3> ret = CorrectPath(secondPoint, firstPoint); //Invert the path

    return ret;
}
```

ANNEXE 3

Fonction permettant de créer un chemin perpendiculaire

```
List<Vector3> CrossWithOtherGuard(GuardTracker guard)
{
    int randomIndex = Random.Range(0, guard.Path.Count - 1);
    int secondIndex = randomIndex + 1;
    float randomTime = Random.Range(0f, 1f);

    Vector3 direction = guard.Path[secondIndex] - guard.Path[randomIndex];
    Vector3 pointInOtherPath = Vector3.Lerp(guard.Path[randomIndex], guard.Path[secondIndex], randomTime);

    Vector3 firstDirection = new Vector3(direction.z, 0, -direction.x).normalized;
    Vector3 secondDirection = new Vector3(-direction.z, 0, direction.x).normalized;

    float distanceMax = Vector3.Distance(guard.Path[0], guard.Path[guard.Path.Count - 1]) / 2f;

    Vector3 firstPoint = pointInOtherPath + (firstDirection * distanceMax * Random.Range(0.9f, 1f));
    Vector3 secondPoint = pointInOtherPath + (secondDirection * distanceMax * Random.Range(0.9f, 1f));


    firstPoint = matrixManager.GetClosestFreePoint(firstPoint, guardPlacementConditions, saveGuardRange);
    secondPoint = matrixManager.GetClosestFreePoint(secondPoint, guardPlacementConditions, saveGuardRange);

    List<Vector3> ret = CorrectPath(firstPoint, secondPoint);

    return ret;
}
```

ANNEXE 4

Fonction permettant de corriger le chemin en cas d'obstacle entre le point de départ et le point de fin

```
List<Vector3> CorrectPath(Vector3 firstPoint, Vector3 secondPoint)
{
    if(!Physics.Linecast(firstPoint, secondPoint, layerMask))
    {
        return new List<Vector3> { firstPoint, secondPoint };
    }
    Vector2 startIndex = matrixManager.ConvertCoordsToIndex(firstPoint);
    Vector2 endIndex = matrixManager.ConvertCoordsToIndex(secondPoint);

    List<Vector3> path = A_Star_PathFinder.SolveA_Star(startIndex, endIndex, guardPlacementConditions);
    if (path.Count == 0) return new List<Vector3>();

    path = SimplifyPath(path);

    path = matrixManager.GetTruePathFromIndex(path);

    return path;
}
```