IA41

INTELLIGENCE ARTIFICIELLE

RAPPORT PROJET

DELIRIUM 2

REALISATION D’UNE INTELLIGENCE ARTIFICIELLE DANS UN JEU VIDEO

ZIANE CLEMENT

NKOUETE-GNOUKOUO WILLY

DUONG TONY

SOMMAIRE

[I. DESCRIPTION DU PROJET 3](#_Toc421797109)

[II. PHASE D’ETUDE 4](#_Toc421797110)

[1. Analyse de l’environnement 4](#_Toc421797111)

[2. Problématique 4](#_Toc421797112)

[3. Recherche du chemin le plus court 4](#_Toc421797113)

[4. Éviter les obstacles 5](#_Toc421797114)

[III. SOLUTION RETENUE 7](#_Toc421797115)

[1. Recherche des diamants 7](#_Toc421797116)

[2. Recherche du chemin le plus court 9](#_Toc421797117)

[3. Gestion des monstres 11](#_Toc421797118)

[4. Gestion des chutes de rochers 13](#_Toc421797119)

[5. Détection de danger imminent 15](#_Toc421797120)

[IV. SCENARIOS 18](#_Toc421797121)

[1. Diamants 18](#_Toc421797122)

[2. Le chemin le plus court 19](#_Toc421797123)

# DESCRIPTION DU PROJET

Dans le cadre de notre UV IA41 à l'UTBM, nous devions réaliser un projet de fin de semestre. Le choix du sujet s'est porté sur la réalisation d'une intelligence artificielle pour le jeu vidéo Delirium 2. Dans ce jeu, le mineur se déplace dans des galeries souterraines à la recherche de diamants. Sur son parcours le mineur doit éviter un certain nombre de pièges, comme des monstres ou des rochers pouvant lui tomber sur la tête. Pour chaque souterrain, le joueur doit ramasser un certain nombre de diamants sur la carte pour ensuite se rendre à la sortie du niveau et passer au souterrain suivant. Le but du projet est donc de rendre le mineur complètement autonome, ce dernier devant trouver seul son chemin pour récupérer les diamants nécessaires, éviter les pièges et les ennemis qui l'entourent puis rejoindre la sortie. Ainsi, toutes les situations devront être étudiées pour que le mineur réponde intelligemment à celles-ci et fasse les choix les plus judicieux pour son évolution.

Nous allons présenter, à travers ce rapport, l’étude menée préalablement  et les différentes solutions que nous envisagions. Par la suite, nous détaillerons la solution que nous avons choisi d’implémenter. Ensuite nous illustreront cette solution avec des jeux d’essais. Enfin nous soulèverons les différents problèmes rencontrés et la façon dont nous les avons traités.

# PHASE D’ETUDE

Dans cette partie, nous allons détailler la phase de réflexion qui a précédé le développement et en particulier quatre points : comment nous avons compris le sujet, la problématique qui en découle, et les deux solutions nécessaires.

## Analyse de l’environnement

La première phase de notre projet fut de comprendre le fonctionnement de l’environnement qui nous a permis de développer notre agent d’intelligence artificielle. En effet, nous avons passé la première semaine à comprendre comment le fonctionnement de Ipseity. Nous avons ensuite appris comment générer de nouvelles cartes afin de pouvoir effectuer des jeux d’essais, comment modifier le périmètre de vision de notre agent pour optimiser notre système. Et enfin nous avons analysé comment les fonctions Prolog étaient appelées par l’environnement et avons constaté qu’elles sont appelées à chaque tour et non de façon récursive ce qui nous a permis d’adapter notre conception.

## Problématique

Dans un deuxième temps, nous avons extrait de notre analyse du sujet et de l’environnement la problématique guidant la conception de notre intelligence artificielle. Nous avons découpé cette problématique en deux parties que nous pouvons synthétiser avec les deux questions suivantes : “Comment trouver le plus court chemin pour récolter les diamants et atteindre le niveau supérieur ?” et “Comment anticiper les obstacles afin d’assurer à notre agent un chemin sans danger ?”.

## Recherche du chemin le plus court

Nous avons réfléchi aux divers solutions possibles pour répondre à la première problématique et nous avons découpé un cheminement en trois étapes, la première étant de trouver la position des diamants les plus proches de notre agent, ensuite il faut générer le plus court chemin jusqu’à ces différentes positions, et une fois que le  nombre de diamant récolté est suffisant pour monter de niveau, nous devons trouver le plus court chemin jusqu’à la porte de fin de niveau.

Tout d’abord nous nous sommes accordés à dire que, pour la première étape, une simple analyse du périmètre de vue était suffisante. Puis une fois l’analyse terminée, nous stockons les différentes positions dans une liste à part pour un traitement ultérieur. Nous avons également anticipé dans notre réflexion que le périmètre de vue de notre agent pouvait être limité et qu’aucun diamant n’apparaissent dedans.

Deuxièmement, nous devions trouver une solution à la recherche du plus court chemin, nous avions analysé l’environnement comme un graphe fini et nous avons décidé d’utiliser un algorithme de recherche du plus court chemin, de plus lors de nos cours nous avions étudié l’algorithme A\* qui nous semblait parfaitement adapté à ce genre de situation.

Enfin nous avions pensé à répéter ces deux premières étapes à pour trouver et acheminer notre agent jusqu’à la porte de fin de niveau une fois que  le nombre de diamants récoltés était suffisant.

## Éviter les obstacles

Dans un deuxième temps, notre réflexion s’est portée sur la sécurité de notre agent durant son parcours du niveau.

Tout d’abord, nous avons répertorié les différentes difficultés que pouvait rencontrer notre agent en deux catégories, les obstacles mobiles tels que les monstres, les chutes de pierres et de diamantes et les obstacles statiques comme les pierres.

Ensuite nous avons commencé par réfléchir aux différentes manières d’éviter à notre agent une rencontre avec un monstre. Nous avions pu extraire de cette réflexion deux solutions envisageable. D’une part nous pouvions anticiper le déplacement des monstres en analysant  leur algorithme de  déplacement et ainsi générer un chemin sécurisé pour notre IA.

Enfin nous avons pensé à une solution pour éviter les obstacles pseudo-mobile, en effet les pierres peuvent passer d’un état statique à un état mobile et vice-versa, nous avons pensé à analyser à différents itérations du jeu, la position des pierres autour pour déterminer si elles tombent ou non, et esquiver les pierres dans le cas où celles-ci seraient en mouvement et à proximité de notre agent.

# SOLUTIONS RETENUES

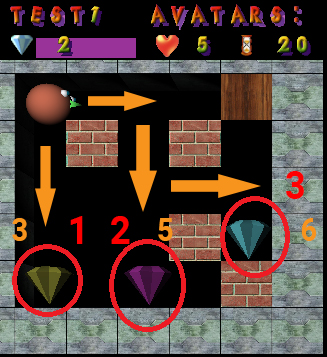
## Recherche des diamants

Nous avons commencé à implémenter notre solution en développant une méthode permettant de retrouver les diamants les plus proches de notre agent (*diamant.pl*).

Tout d’abord nous vérifions qu’il y a bien des diamants dans le périmètre de vue de notre mineur, le périmètre de vue étant par défaut toute la carte du jeu. Si des diamants sont visibles par le mineur et accessibles, nous stockons leur position dans une liste. Cette liste sera analysée afin de trouver le diamant le plus proche. Pour chaque diamant, nous utilisons l’algorithme A\* et gardons en mémoire la longueur du chemin. Cette longueur de chemin sera comparée entre tous les diamants et ainsi, nous pouvons trouver le diamant le plus proche.

La position de ce plus proche diamant sera utilisée par l’algorithme de recherche du plus court chemin (*astar.pl*) comme étant la prochaine destination de notre personnage.

Nous recommençons cette étape à chaque itération du jeu jusqu’à ce que notre mineur ait récolté assez de diamant pour pouvoir chercher la sortie et ainsi passer au niveau supérieur.

Dans cette figure, vous pouvez remarquer la longueur du chemin (*en orange*) ainsi que la priorité (*en rouge*) pour aller vers chaque diamant.

Dans ce cas particulier, il va prendre le chemin vers le diamant jaune (de longueur 3).

## Recherche du chemin le plus court

A\* est un algorithme reconnu de recherche du plus court chemin dans des graphes.

Il a été implémenté pour gérer les déplacements du mineur à partir d'un point de départ donné, d'un point d'arrivée et d'une liste représentant la carte du jeu (*astar.pl*).

A partir de ces éléments, l'algorithme étudie les directions possibles que peut prendre le mineur.

Voici l’algorithme détaillé de l’A\* :

1. Ajouter la position de départ du monstre dans la liste ouverte avec un coût de 0.
2. Répétition :
   * Chercher parmi les quatre cases adjacentes à la case en cours celle dont le coût est le plus petit. Cette case devient la case actuelle.
   * On ajoute la position de la case en cours à la liste fermée (elle est considérée comme analysée).
   * Pour chacune des 4 cases adjacentes de la case en cours, nous faisons les tests suivants.
     1. Si c’est une case non traversable (rochers, murs) ou si c’est une case de la liste de la liste fermée, alors cette case est ignorée.
     2. Sinon on l’ajoute dans la liste ouverte. Cette case fera partie d’une des cases fils de la case actuelle. Calculer le coût de cette case.
   * Nous trions la liste de manière croissante en fonction du coût.
3. Arrêt :
   * Si la destination finale a été trouvée, alors le chemin à partir de la case de départ vers cette case de destination est sauvegarde. Dans ce cas, le mineur prendra ce chemin pour aller vers la destination.
   * Sinon si la liste ouverte est vide, cela signifie qu’un chemin n’a pas été trouvé. Dans ce cas, la liste définissant le chemin sera vide. Le mineur aura deux comportements possibles à cet état : soit un danger imminent est détecté (l’approche d’un monstre à proximité lors de la prochaine itération, ou un rocher en chute se situant juste au-dessus de celui-ci) auquel cas il cherchera à se placer en position de sécurité ; soit aucun danger n’est détecté auquel cas il ne bougera pas pendant cette itération du jeu.

## Gestion des monstres

Durant son évolution, le mineur devra éviter différents monstres qui cherchent à le manger. Notre mineur meurt s’il se situe sur l’une des quatre cases adjacentes d’un monstre. Nous avons dû trouver une astuce intelligente afin d’assurer à notre personnage la possibilité de récolter les diamants nécessaires et d’atteindre la sortie sans se faire manger par les monstres.

Tout d’abord, nous stockons la carte à un instant *t-1* pour la comparer à un instant *t*  afin de prévoir le sens du déplacement des monstres. Une fois le sens du déplacement des monstres prévus, nous créons une liste temporaire qui serait la carte à l’instant *t+1* avec les futures positions des monstres. Nous anticipons ainsi leur déplacement.

Les cases autour d’un monstre dans cette carte seront considérées comme étant une case impossible à atteindre et ainsi notre mineur ne la prend pas en compte dans son chemin et s’assure un chemin sans danger pour atteindre sa cible *(monster.pl).*

Voici l’algorithme des monstres :

* Si c’est la première itération du jeu, la direction prise sera dans l’ordre {Haut, Droite, Bas, Gauche, NeBougePas}. *(s’il ne peut pas aller en haut, il cherchera à aller à droite, s’il ne peut pas aller à droite, il cherchera à aller en bas, etc…)*
* Sinon la direction prise sera en fonction de sa direction précédente. Le monstre cherchera à aller à sa gauche en premier lieu, puis *en cas d’obstacle* à continuer dans sa direction actuelle, puis à aller à sa droite, puis à faire demi-tour, puis *dans le cas où il y a des obstacles partout autour de lui* il ne bouge pas).

Cet algorithme nous a permis d’améliorer l’efficacité de notre mineur.

Voici un exemple de déplacement d’un monstre lorsqu’il n’y a pas d’obstacles.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *t* | *t+1* | *t+2* | *t+3* | *t+4* |
| C:\Users\Tony\AppData\Local\Microsoft\Windows\INetCache\Content.Word\iamonstre.png | C:\Users\Tony\AppData\Local\Microsoft\Windows\INetCache\Content.Word\iamonstre2.png | C:\Users\Tony\AppData\Local\Microsoft\Windows\INetCache\Content.Word\iamonstre3.png | C:\Users\Tony\AppData\Local\Microsoft\Windows\INetCache\Content.Word\iamonstre4.png | C:\Users\Tony\AppData\Local\Microsoft\Windows\INetCache\Content.Word\iamonstre5.png |

Voici un exemple de carte prédictive. A partir de la carte au temps *t-1* et celle au temps, nous pouvons déterminer la prochaine direction du monstre, et par la même occasion, sa prochaine case de destination. Les carrés rouges représentent des obstacles virtuels sur la carte modifiée. Le mineur considérera, lors de la recherche de chemin, ces cases comme des obstacles. Il n’y a donc aucune chance qu’il meurt à cause des monstres.

|  |  |  |
| --- | --- | --- |
| *t-1* | *t* | *t+1* (carte prédite) |
| C:\Users\Tony\AppData\Local\Microsoft\Windows\INetCache\Content.Word\iamonstre.png | C:\Users\Tony\AppData\Local\Microsoft\Windows\INetCache\Content.Word\iamonstre2.png | C:\Users\Tony\Pictures\iamonstrepredi.jpg |

## Gestion des chutes de rochers

Le deuxième type d’obstacles rencontrés par le mineur est la chute de pierres. En effet, lorsque le mineur creuse, il peut laisser un vide sous certaines pierres et faire chuter celles-ci.

Afin d’éviter la mort du mineur, nous avons implémenté une solution relativement proche de celle qui permet d’esquiver les monstres.

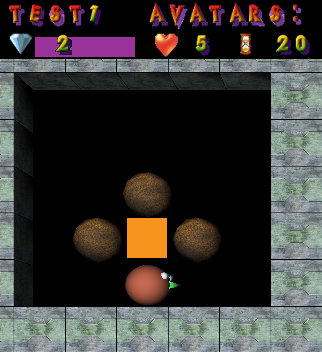
Tout d’abord nous comparons la position des pierres à un instant *t-1* et *t* pour déterminer si celles- ci sont en mouvement, si c’est le cas nous bloquons les cases susceptibles d’être atteintes par un rocher et de tuer notre mineur. Ces cases seront considérées comme des obstacles dans la carte de prédiction et l’algorithme A\* les considèrera comme des cases non atteignables pour la recherche du plus court chemin.

Afin de déterminer le danger que représente un rocher, nous avons sauvegardé la liste répertoriant tous les rochers ainsi que leur état respectif (en cours de chute ou état stable).

Voici les cases qui seraient bloquées à l’instant *t+i* dans cette configuration. Les cases en orange représentent les obstacles virtuels sur la nouvelle carte modifiée.

|  |  |  |  |
| --- | --- | --- | --- |
| t | *t+1* | *t+2* | *t+3* |
| C:\Users\Tony\AppData\Local\Microsoft\Windows\INetCache\Content.Word\iarocherdiam.jpg | **C:\Users\Tony\Pictures\iarocherdiam2.jpg** | C:\Users\Tony\Pictures\iarocherdiam2.png | C:\Users\Tony\Pictures\iarocher5.png |

**Cas particulier**

Sur cette figure, il n’y a pas d’obstacles virtuels à gauche et à droite du mineur (même si le rocher est en cours de chute).

Supposons que le mineur soit en position *(x,y)* alors les rochers qui sont en position *(x+1,y+1)* et *(x-1,y+1)* ne représente pas de danger pour lui. Comme sur l’illustration précédente, il pourra choisir de passer à gauche ou à droite.

## Détection de danger imminent

En plus de la gestion des monstres et des chutes de rocher qui nous ont permis de créer une carte prédictive, nous avons anticipé le fait que des dangers puissent survenir lorsque le échoue à trouver un chemin vers sa destination (*auquel cas le choix par défaut serait de ne pas bouger et de recommencer toute la procédure à l’itération suivante pour trouver un diamant ou une sortie tout en gérant les dangers et obstacles*) avec le prédicat *haveToEscape.*

Dans ce cas-là nous testons deux types de danger potentiel :

* + La chute des rochers sur les deux cases au-dessus du mineur (*fromFallingThings*)
  + les monstres qui pourraient arriver sur l’une des cases adjacentes au mineur (*fromMonsters*)

Le prédicat *haveToEscape* renvoie vrai si l’un des deux prédicats *fromFallingThings ou fromMonsters* est vrai. Dans ce cas, le mineur cherchera la première case adjacente disponible (*toutes les cases disponibles de la carte modifiée par monster.pl et rock.pl sont sécurisées et aucun danger ne peut survenir à ces positions*).

Le mineur se déplacera vers cette position.

Danger imminent de rochers

|  |  |  |  |
| --- | --- | --- | --- |
| *t* | *t+1* | *t+2* | *t+3* |
|  |  |  | C:\Users\Tony\Pictures\iarocher5.png |

Ici, contrairement au cas précédent, il n’y a pas de diamants sur la carte. Le mineur par défaut choisirait donc de rester sur place. Néanmoins, il doit d’abord passer par la *détection de danger imminent.*

* Au temps *t+1*, il détecte le rocher qui est deux cases au- dessus de lui. S’il ne se déplace pas, il mourrait à la prochaine itération. Il doit donc choisir de se déplacer et ainsi se placer en position de sécurité.
* C’est également le cas au temps *t+2,* où le rocher se situant juste au-dessus de lui représente un danger imminent. Il choisit donc encore de se déplacer.

A travers cet exemple, nous avons couvert les deux cas où un danger imminent de rocher pourrait survenir.

Danger imminent de monstres

Ici, le diamant est inaccessible. Il passe donc par la *détection de danger imminent* comme pour le cas précédent.

* Au temps t, aucun danger détecté. Il ne bouge pas.
* Au temps t+1, un monstre arriverait au temps t+2 à proximité. Il choisit de fuir et de se mettre en position de sécurité.
* Etant donné la configuration de la carte, il va se mettre ensuite en état de fuite indéfiniment.

|  |  |  |
| --- | --- | --- |
| *t* | *t+1* | *t+2* |
| C:\Users\Tony\Pictures\iadangi.png | **C:\Users\Tony\Pictures\iadangi2.png** | C:\Users\Tony\Pictures\iadangi3.png |

# SCENARIO

# PROBLEMES RENCONTRES

Lors de la réalisation de notre produit, nous nous sommes confrontés à quelques problèmes.

* Le problème de périmètre de vision. Au départ, le périmètre de vision étant la carte entière

# AMELIORATIONS POSSIBLES

Dans notre projet, nous avons pu couvrir la majorité de nos objectifs. Malgré cela, l’intelligence artificielle de notre robot reste améliorable.

* Nous ne gérons qu’un seul type de monstre : le monstre rouge. Nous n’avons pas géré les déplacements des monstres bleus (ils ont un déplacement similaire aux monstres rouges mais font une « pause » tous les deux itérations.)
* Nous pourrions améliorer l’intelligence de notre robot en lui permettant d’être capable de tendre des pièges en se servant des rochers.
* Lorsqu’aucun chemin n’est trouvé vers sa destination (diamant ou sortie inaccessible), il pourrait pousser les rochers qui gêneraient éventuellement son itinéraire.

* Notre mineur ne sait pour l’instant prévoir une situation à l’instant *t+1.* Nous pourrions le doter d’une capacité de prédiction supérieure et ainsi l’éviter de se retrouver dans des positions dangereuses et « bloquantes ». Ceci est possible car on peut anticiper les déplacements des monstres ainsi que les chutes de rocher.
* Nous n’avons pas modifié le périmètre de vue de notre mineur. Nous l’avons laissé par défaut (toute la carte) et par conséquent, nous nous retrouvons avec un débordement de pile sur des cartes très grandes (40\*24 par exemple). La première solution serait d’adapter notre périmètre de vue en fonction de la situation (de base, il serait très petit et augmenterait jusqu’à ce qu’un diamant soit dans son périmètre de vue). La deuxième solution, plus ardue, serait d’améliorer notre algorithme de recherche de diamant.

# CONCLUSION