Baptiste BRINON **Nicolas MARTINEZ**

Rapport de projet
Fuite de Criminels
25 Mai 2020

L3 Informatique **Université d'Angers**

SOMMAIRE

INTRODUCTION

Ce projet à été réalisé par Nicolas Martinez et Baptiste Brinon dans le cadre de la fin de notre 3ème année de licence informatique à l'université d'Angers. Pour valider notre année, nous devions choisir dans une liste de sujets celui que nous allions réaliser durant les 6 semaines suivantes. Ainsi, ayant tous les deux choisis des options concernant l'IA, nous avons choisi le sujet qui nous permettrait de plonger dans ce domaine qui nous passionne tous les deux : « *Fuite de criminels* » encadré par Florian Delaverhne**.**

Le principe est tel que suit : il y a, sur un terrain, des gendarmes et des voleurs ainsi que des sorties. Les gendarmes doivent attraper les voleurs, et les voleurs doivent réussir à atteindre les sorties sans se faire attraper par les gendarmes.

Nous apprécions particulièrement le fait de devoir concevoir une IA pour un camp, puis trouver comment la contrer pour l'IA du camp opposé, nous forçant ainsi à constamment aller plus loin dans l'élaboration de nos IA.

De plus, le peu d'importance accordé à l'interface graphique de ce sujet a permis de limiter le principal soucis de notre binôme : ayant tous deux choisis les options d'IA ce semestre, nous n'avons que des connaissances basiques en GUI. Ce sujet ne demandant qu'une visualisation simple du terrain, ça nous a permis de ne pas être pénalisés par nos choix d'options.

I – Mise en place du Projet

1 Outils utilisés

Le Langage de programmation utilisé est le C++. Nous avions plusieurs choix possibles, mais nous avons décidé de réaliser le projet en C++ car c'est le langage que nous avons le plus utilisé durant cette année. Il s'agissait d'une opportunité supplémentaire d'utiliser ce que l'on a vu en cours dans un contexte extra-scolaire.

Qt a été utilisé pour la partie graphique. Nous n'avons utilisé que peu de ses possibilités, mais suffisamment pour avoir un affichage satisfaisant et correspondant à ce qui était demandé. Qt a été le choix par défaut, car il s'agit de la seule API graphique que l'on a vu en cours et avec laquelle nous avions donc déjà eu un premier contact.

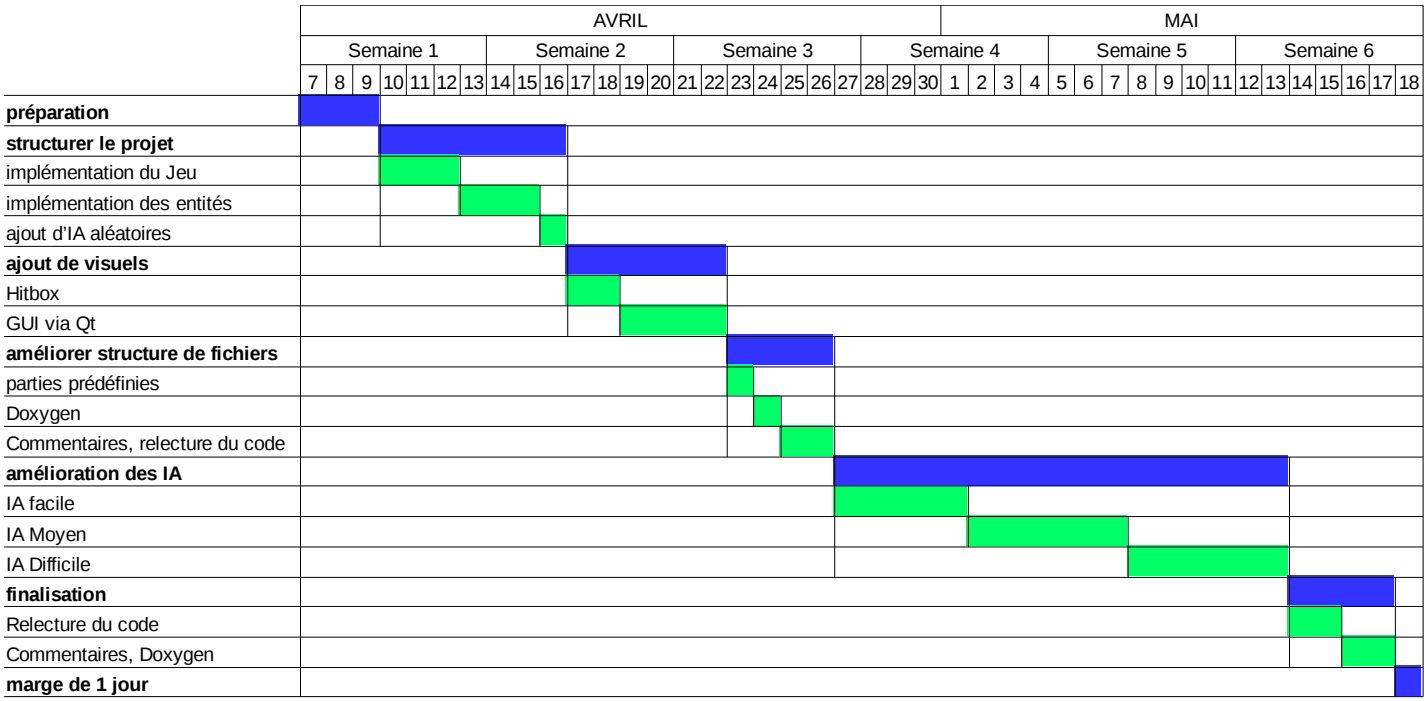
Nous avons utilisé un dépôt git pour la mise en commun du code, ce qui c'est révélé très pratique durant le confinement. En plus de la mise en commun du code, github offre des fonctionnalités comme la possibilité de récupérer une ancienne version du code, de suivre les modifications qui ont été effectuées, et de suivre l'investissement de chacun des participants.

Pour compiler le C++ et QT, nous avons utilisés Cmake. Cependant, il prends de la place (notamment via les moocs) dans la structure de fichiers, et nous avons donc réalisé un script Shell pour la compilation. Ce script fait le build en utilisant Cmake, compile, déplace l'exécutable puis supprime les fichiers superflus créés par Cmake. La compilation est de ce fait un peu plus longue, mais a moins d'impact sur l'organisation des fichiers.

2 Planning

2.1 Planning prévisionnel

Nous n'avions, au début, aucune idée de la date à laquelle nous devrions rendre le projet à cause du confinement qui a désorganisé le calendrier annuel initial. Cependant, des enseignants émettaient l'hypothèse que nous n'aurions que six semaines pour réaliser le projet. Nous sommes donc partis de cette hypothèse pour la répartition de différentes étapes sur le temps qui nous était imparti.

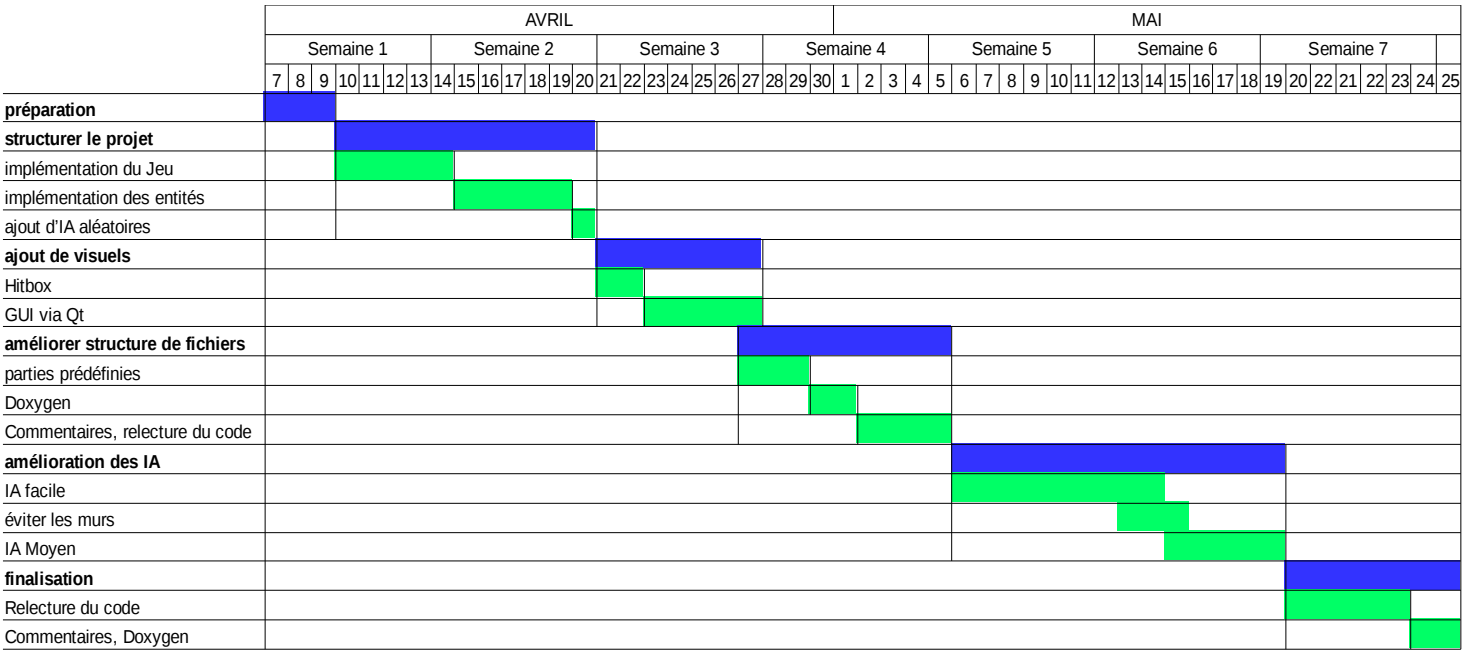


Esquisse de diagramme de Gantt correspondant à l'emploi du temps prévu

2.2 Réel emploi du temps

Cependant, il y eut quelques imprévus. Lors de la réalisation du corps du programme, les idées d'implémentation d'IA nous ont fait apporter de nombreuses modifications à l'idée que nous en avions initialement, et l'implémentation d'esquive de murs nous a fait revoir tout le fonctionnement des IA. Tout ceci nous pris donc plus de temps que prévu.

Nous avions en fait sept semaines et non six, et au dernier moment nous avons eu trois jours en plus. Hors, même avec ces trois jours supplémentaires nous n'avons pas eu le temps de réaliser une IA Difficile.



Esquisse de diagramme de Gantt correspondant à l'emploi du temps réel

3 Répartition des tâches

Baptiste Brinon : Implémentation du Jeu (50%), Implémentation des entités (70%), parties prédéfinies (70%),Doxygen,Commentaires(90%) et Relecture du code, IA Facile(75%),éviter les murs, IA Moyen, relecture du code, Commentaire et Doxygen bis.

Nicolas Martinez : Implémentation du Jeu (50%), Implémentation des entités (30%), Ajout d'IA aléatoire, Hitbox, GUI, parties prédéfinies (30%), Commentaires(10%), IA Facile (25%).

La répartition des tâches ci-dessus n'est pas une planification, mais le résultat final du déroulement du projet. Pour organiser les tâches restantes et garder une trace des tâches réalisées, nous avons plutôt utilisé un fichier dédié. Il s'agit de RESTE_A_FAIRE.txt, dont le contenu est de la forme suivante :

```
21  OK  - ajout de sortie sur fichier texte sous forme d'historique
22  X   - suppressions des pointeurs (-> entraine utilisation de pointeurs sur le vecteur plutôt que vecteur de pointeurs)
23  X           après une tentative avec des pointeurs, et une autre avec un passage par référence, aucun des deux ne fonctionnent.
24      - implémenter des "valeurs par défaut (randoms?)" dans les constructeurs
```

Extrait du fichier RESTE_A_FAIRE.txt

Concernant l'extrait ci-dessus :

- La mention « OK » indique que la tâche a bien été réalisée.
- La mention « X » indique que la tâche n'est pas réalisable, ou qu'elle n'est plus intéressante suite à un autre changement.
- Les tâches sans mentions sont des tâches qui n'ont pas encore été traitées, et qui doivent encore être réalisées.
- une lignée indentée correspond à des détails supplémentaires concernant la ligne supérieure. Elle hérite de ce fait de sa mention.

II - Structure du projet et concepts

1 Les différentes fonctionnalités « de base »

Les différentes fonctionnalités dites de bases correspondent aux premières esquisses de notre réflexion sur le sujet, de quoi nous avions besoin pour répondre aux attentes. Ces fonctionnalités ont été amené à changer au cours du projet, des traces sont présentes dans le dossier archive.

Nous reviendrons sur ces différences dans la partie *V – Déroulement du projet*.

1.1 L'interface graphique

Pour réaliser le projet et avoir un rendu graphique nous avions besoin d'une GUI, en l'occurrence notre choix s'est porté sur QT. Nous avions eu l'occasion d'utiliser QT au premier semestre dans le cadre du cours d'algorithmique pour réaliser une calculatrice, l'interface étant assez simple nous n'avions pas vu comment rendre l'affichage «dynamique» et aucun de nous deux n'avait suivi l'option sur les interfaces graphiques.

A partir de la documentation disponible nous avons pu créer dans un premier temps une fenêtre de configuration de partie et ensuite une fenêtre de jeu où l'on retrouve nos entités sous forme de rectangles permettant de visualiser le déroulement de la partie après le lancement de celle-ci.

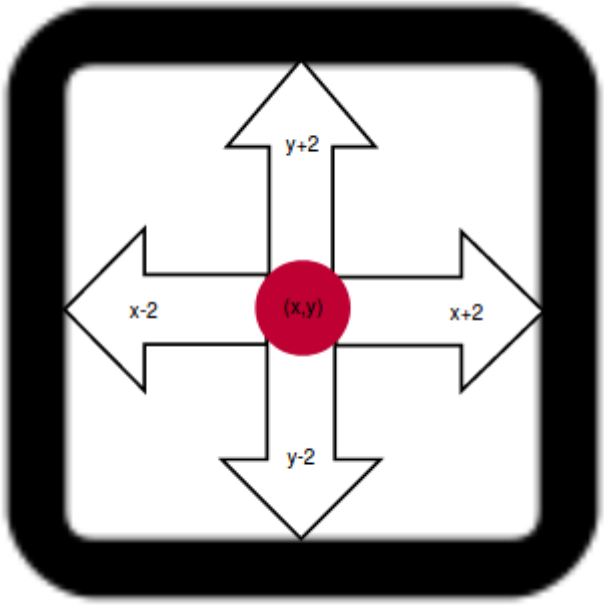
Nous reviendrons sur son fonctionnement en partie *III – Fonctionnement – Partie GUI*

Le GUI étant notre retour graphique sur les différents déplacements des personnages c'était notre seconde priorité après l'intégration du jeu.

1.2 Une hitbox

Les différentes entités étant représentées par des carrés sur l'interface, il nous fallait définir une hitbox. En effet, ces entités n'avaient que des Positions sous la forme de coordonnées (x,y). Hors, elles devaient avoir une certaine "largeur" pour pouvoir se toucher mutuellement, d'où l'ajout d'une hitbox. La dimension de cette hitbox est définie par une constante globale.

Exemple avec une taille HITBOX de 4 positionnée selon le centre du carré.



Représentation d'une Hitbox de taille 4, le point rouge correspond à la position de l'entité

Cette hitbox se retrouve donc ensuite positionnée sur la position du joueur, ce qui permet aux gendarmes de capturer un voleur et aux voleurs de s'échapper par une sortie en cas de contact.

1.3 Déplacement

Les déplacements sont représentés par des vecteurs pointant sur une position d'arrivée. Ces vecteurs sont le résultat d'une direction dans laquelle se dirige le personnage, sur laquelle on applique une vitesse.

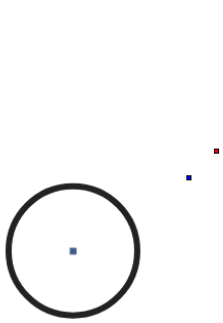
Cette vitesse est attribuée à la construction du joueur.

A partir du calcul de déplacement nous vérifions que le mouvement ne va pas amener le joueur à sortir hors du terrain et on évite les vecteurs de trop longue distance afin que les déplacements ne soient pas "monotones".

1.4 Champ de vision

Pour permettre à nos joueurs de se déplacer de façon plus réfléchie lors de l'utilisation des IA il nous fallait définir un champ de vision ou portée de vue. Dans un premier temps elle était explicitement définie pour chaque joueur en fonction de la situation. Un voleur avait un champ de vision réduit comparé aux gendarmes qui pouvaient repérer le voleur beaucoup plus tôt et ainsi commencer sa chasse. Nous avons finis par définir une variable globale correspondant à une distance de 50, les joueurs ont donc la même vision mais n'en ont pas la même utilisation.

Cette portée de vue correspond à la distance euclidienne entre deux entités. En fonction des différentes IA elle peut être amenée à être utilisée comme unité de mesure.



Représentation du champ de vision pour un joueur

1.5 Déroulement d'une partie

La partie débute au moment où la fenêtre de jeu apparaît. Un chronomètre démarre en arrière-plan afin d'avoir la durée de la partie et compléter notre fichier de score contenant l'historique des parties.

Le déroulement d'une partie est composé de plusieurs tours, ceux-ci s'exécutent tant que des voleurs sont présents sur le terrain. On calcule dans un premier temps les déplacements de nos joueurs, nous les appliquons ensuite et pour chaque gendarme nous vérifions si il capture un voleur, c'est à dire si leurs hitbox se touchent. Dans le cas du voleur il se passe exactement la même chose avec les sorties, si la hitbox du voleur vient à être en contact avec celle d'une sortie alors le voleur s'échappe et ne se trouve alors plus présent dans la liste des voleurs jouant la partie (Même chose en cas de capture, on supprime le voleur de la liste).

Dans le terminal nous retrouvons toute information sur le déroulement de celle-ci à savoir la position de toutes les entités ainsi que leurs hitbox à chaque tour. Dans le cas du voleur nous affichons aussi sa distance actuelle par rapport à la sortie la plus proche ainsi que la distance après déplacement afin de ne pas éviter de s'approcher du mur si la sortie est proche de celui-ci. Nous affichons également ce que nous appelons "ping" comme étant la position actuelle du joueur + son vecteur de déplacement afin de mieux visualiser leur mouvement. Si tout se passe correctement, le déplacement total correspond à la vitesse du joueur.

```
 TOUR 36 :
dist actuelle :23,446165vs dist  destnation :22,736436
PING V(-17,248174,15,881535) + (1,112489,-0,887511) ---> (-16,135684,14,994024) déplacement total:2,000000
PING G(-25,221214,21,942628) + (1,133274,-0,866726) ---> (-24,087941,21,075902) déplacement total:2,000000
PING G(-4,439882,6,485119) + (-1,157730,0,842270) ---> (-5,597613,7,327389) déplacement total:2,000000

Liste Voleurs avec position et Hitbox :
V1 position : (-16,135684,14,994024) | Hitbox:(-18,135684,-14,135684,16,994024,12,994024)

Liste Gendarmes avec position et Hitbox :
G1 position : (-24,087941,21,075902) | Hitbox:(-26,087941,-22,087941,23,075902,19,075902)
G2 position : (-5,597613,7,327389) | Hitbox:(-7,597613,-3,597613,9,327389,5,327389)

Liste NonJoueur avec position et Hitbox :
Sortie position : (0,000000,0,000000) | Hitbox:(-2,000000,2,000000,2,000000,-2,000000)

 TOUR 37 :
dist actuelle :22,026826vs dist  destnation :21,317428
PING V(-16,135684,14,994024) + (1,109124,-0,890876) ---> (-15,026561,14,103148) déplacement total:2,000000
PING G(-24,087941,21,075902) + (1,130261,-0,869739) ---> (-22,957680,20,206163) déplacement total:2,000000
PING G(-5,597613,7,327389) + (-1,163729,0,836271) ---> (-6,761342,8,163659) déplacement total:2,000000

Liste Voleurs avec position et Hitbox :
V1 position : (-15,026561,14,103148) | Hitbox:(-17,026561,-13,026561,16,103148,12,103148)

Liste Gendarmes avec position et Hitbox :
G1 position : (-22,957680,20,206163) | Hitbox:(-24,957680,-20,957680,22,206163,18,206163)
G2 position : (-6,761342,8,163659) | Hitbox:(-8,761342,-4,761342,10,163659,6,163659)

Liste NonJoueur avec position et Hitbox :
Sortie position : (0,000000,0,000000) | Hitbox:(-2,000000,2,000000,2,000000,-2,000000)
```

Exemple d'affichage d'informations sur un tour dans le terminal

1.6 Définir une fin de partie

Nous avions pour idées d'implémenter différentes fins de partie, telles qu'un score à obtenir au bout d'une certaine durée, ou bien une partie "tout ou rien" dans laquelle tous les voleurs doivent s'échapper, ou encore dans le même genre, une partie où les Gendarmes doivent attraper tous les voleurs.

Cependant, nous avons gardé une fin de partie plus classique: la partie se termine quand il ne reste plus de voleurs sur le terrain. Suite à ça, on établi des statistiques (nombre de voleurs échappés, nombre de voleurs capturés,...) qui sont enregistrées dans un fichier texte conservant une trace des parties jouées.

2 Les fonctionnalités "bonus"

Par la suite nous avons implémenté différentes fonctionnalités que nous qualifions de « bonus » pour faciliter l'utilisation de l'interface ou encore avoir un retour sur les parties exécutées. Ces différentes idées sont venues au fur et à mesure de l'avancé de notre projet.

2.1 Ajouts de différents boutons dans la fenêtre de configuration

Dans un premier temps, à partir de la fenêtre de configuration nous pouvions uniquement ajouter des entités puis lancer la partie ou alors lancer la partie selon des exemples prédéfinis.

Nous avons donc décidé d'ajouter un bouton d'aperçu qui permet de voir la position initiale de chaque entités avant de lancer la partie, ou même de rouvrir la fenêtre si elle a été fermée par inadvertance. Ensuite, nous avons ajouté un bouton "fin de partie" qui permet d'en terminer l'exécution plus proprement que par la fermeture classique et de ne pas avoir de fuite mémoire, ce qui permet de relancer une partie dans la foulée.

2.2 Exemples prédéfinis

Ces exemples représentent en réalité des tests montrant le fonctionnement des différentes IA implémentées et comment dans une situation un voleur peut s'échapper, ou encore comment un gendarme se positionne pour protéger une sortie.

C'était dans un premier temps un gain de temps considérable, plutôt que d'ajouter manuellement des entités sur le terrain on se retrouve avec des parties pré-configurés permettant, comme dit précédemment, de tester différentes situations où par exemple un voleur se trouve piégé entre deux gendarmes.

2.3 Fichier texte et timer

Pour avoir une trace sur les différentes parties manuelles ou bien sur nos exemples nous avions besoin d'un fichier texte. (cf. Capture d'écran en annexes).

Dans ce fichier texte nous retrouvons la durée des parties grâce à la bibliothèque chrono et différentes informations sur la partie, comme le nombre de voleurs/gendarmes en début de partie et enfin le nombre de voleurs capturés ou sortis. Ce qui correspond finalement au score final de la partie.

Ce fichier nous permet donc d'avoir un suivi sur toutes les parties que nous avons exécutées.

3 Les différentes Intelligences Artificielles

Présentation rapide des différentes IA que nous avons été amené à concevoir (ou envisager) durant le projet. Il s'agit d'une explication des différents niveaux d'IA. Pour l'explication détaillée de leur fonctionnement, allez au chapitre III.

3.1 temporaires

Les premières « IA » conçues n'étaient pas réellement des IA, bien qu'elles étaient à l'emplacement de celles-ci. Il s'agit de déplacements prévisibles servant à faire des tests. Des « IA » telles que « vers_haut » « vers_bas » « suit_voleur » ou même « fuit_chasseur » qui avaient des comportements très prévisibles, permettant ainsi de s'assurer que tout le reste du programme fonctionnait correctement avant de passer aux IA plus complexes.

3.2 IA Aléatoire

Il s'agit d'une IA imprévisible dont les déplacements sont presque aléatoires. Presque, car elle est tout de même soumise aux règles du jeu, comme les autres IA (ne pas sortir du terrain,…).

3.3 IA Facile

Il s'agit de la première IA dont nous avons dû réfléchir au comportement. Elle prends en compte l'ennemi le plus proche pour calculer son déplacement, ainsi que la position de la sortie la plus proche. Le poursuite de voleurs ainsi que la fuite de gendarmes sont très basiques, et il n'y a aucun jeu d'équipe. Cependant, la surveillance des sorties par les gendarmes inactifs en est un prémice.

3.4 IA Moyen

Il s'agit d'une reprise de l'IA Facile, mais en prenant en compte plus d'éléments, et dont les priorités ont été réajustées. En effet, cette fois l'IA prends en compte tous les alliés à portée ainsi que tous les opposants repérés, et non pas juste le plus proche. L'ennemi le plus proche garde cependant un plus grand impact que ceux qui sont éloignés. Les voleurs se sont vus attribués des stratégies d'évitement permettant d'éviter de se faire encercler par des gendarmes, tandis que les gendarmes ont une surveillance des sorties améliorées, essayant de ne pas partir à la poursuite d'un Voleur déjà poursuivi par un collègue.

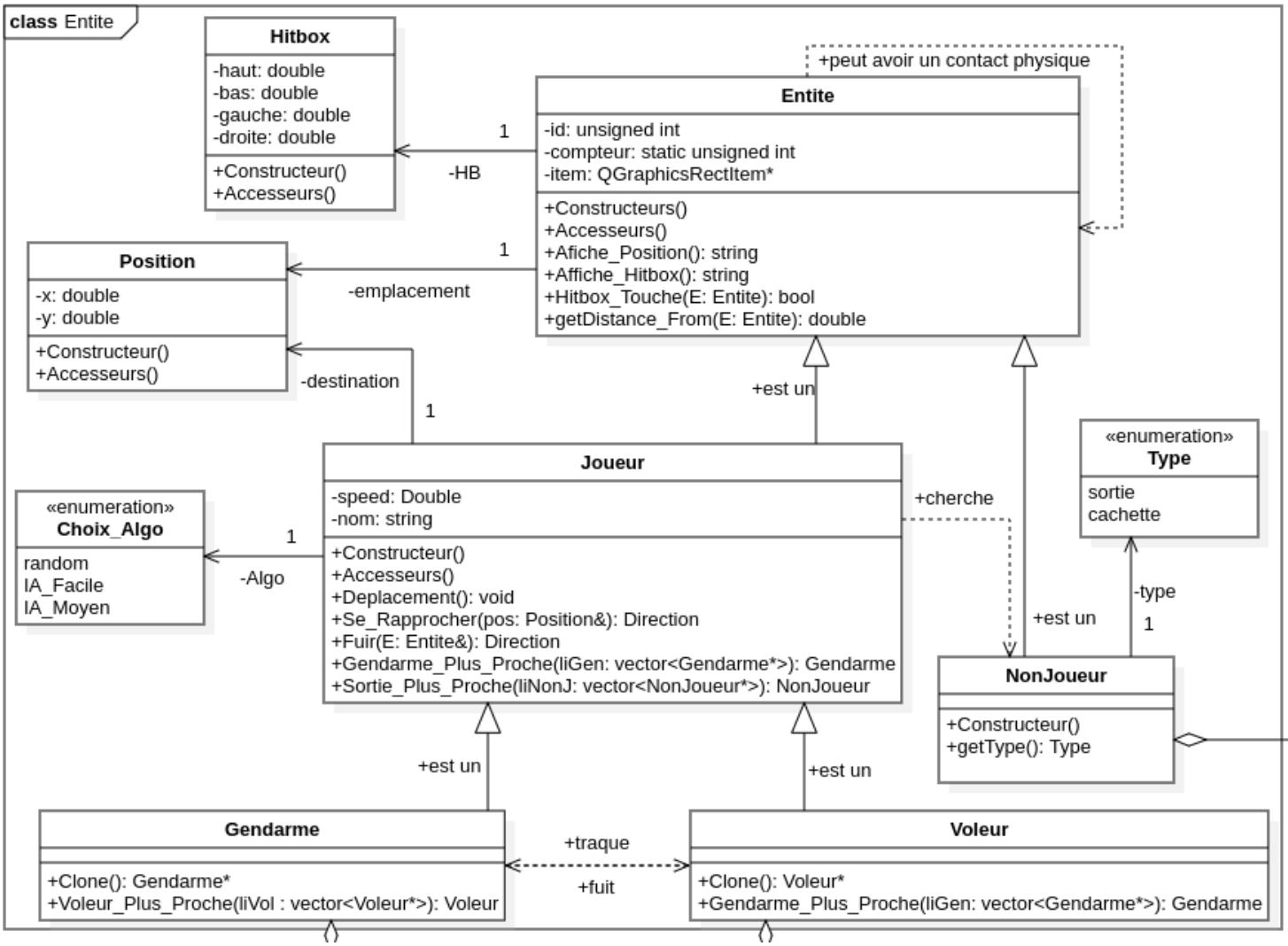
3.5 IA Difficile

Bien que nous n'avons pas eu le temps de réalisé d'IA difficile, nous avions plusieurs idées concernant celles-ci, ainsi que des essais à faire. Cependant, nous aurions sans doutes été encore plus loin, les idées nous venant surtout en observant le comportement des IA quand elles s'affrontent.

4 Structure des classes

4.1 Diagramme de classe

Il peut être divisé en deux parties. La première est dédiée au fichier Entite.hh

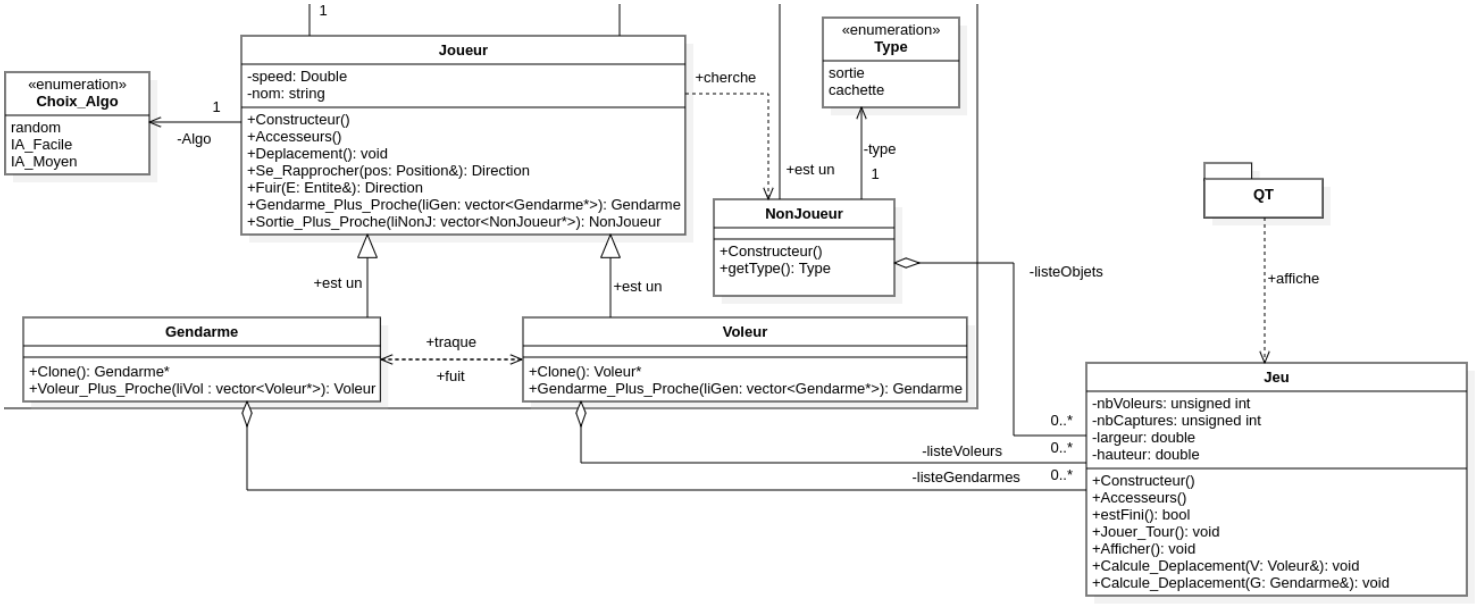


A partir du diagramme de classe on remarque qu'un joueur est une entité, composé d'une position, d'une hitbox et dans son constructeur on peut choisir son algorithme. Ce Joueur est un Voleur ou un Gendarme, ces deux types de joueurs interagissent entre eux pour se traquer/se fuir.

Même chose pour un NonJoueur, c'est une entité composé d'une position et d'une hitbox et dans son constructeur on peut choisir son type à savoir uniquement des sorties dans le cas de notre projet.

Une entité possède un "contact physique" avec les autres entités grâce à sa hitbox.

La deuxième partie est dédiée à la classe Jeu et son lien avec les Entites.



La classe Jeu inclut donc une liste de Voleur(s)/Gendarme(s) et de Sortie(s) et initialise le terrain en fonction des informations données sur les entités. Par la suite QT exécute et affiche le déroulement d'une partie et met à jours toutes les informations dans la fenêtre de jeu à partir du moment où un déplacement est effectué par un joueur.

Une version complète est disponible en annexe.

4.2 Répartition des classes dans les fichiers

Pour une meilleur lisibilité et compréhension de la structure de nos classes nous avons plusieurs fichiers permettant de mieux nous organiser en cas d'ajout/modification de fonctionnalités.

"**Entites"** -> Position, Direction , Hitbox , RectItem et l'affichage de ces informations. Représentation de Joueur/NonJoueur correspondant aux voleurs/gendarmes/sorties.

"**Jeu"** -> Intégration des entités, construction du terrain et initialisation d'une partie. Le calcul des déplacements des joueurs ainsi que le déroulement d'une partie et son affichage dans le terminale.

"**Fenetre_Ajout"** -> Correspond à la fenêtre de configuration. Ajout des Joueurs/NonJoueurs et ajout de ces informations sur la fenêtre de jeu.

"**Fenetre_Jeu"** -> Correspond à la fenêtre où se déroule une partie ainsi que la fenêtre d'aperçu.

"**IA_Gendarme/IA_Voleur"** -> Permet de définir pour chaque joueur les différents algorithmes utilisés pour calculer un déplacement. Nous les avons mis dans un fichier à part pour pouvoir facilement y accéder sans avoir à passer par tout le code de Entites.cc, et pour pouvoir include Jeu.hh plutôt que Entite.hh

"**Exemples"** -> Regroupe les parties pré-configurées. Nous les avons mis dans une fichier dédié pour pouvoir facilement en ajouter et retirer pendant qu'on modifie les IA, permettant d'ajuster différents tests rapidement.

"**Main"** -> Permet d'exécuter la fenêtre de configuration.

III – Fonctionnement

1 Jeu et Personnages

1.1 Jeu

Il fallait, avant tout, définir un cadre de jeu. Un environnement dans lequel les personnages pourraient évoluer. Il fallait donc qu'une classe contienne la liste des éléments du jeu, et qui serait utilisée pour assurer le déroulement de la partie.

Jeu a été créée dans ce but. Nous y avons d'abord placé les entités, ainsi que de quoi les construire et les détruire. Nous y avons aussi placé les dimension du terrain, et de quoi y accéder.

Plus tard, nous lui avons ajouté des méthodes et un attribut permettant d'obtenir des statistiques. Jeu retiens donc aussi le nombre total de voleurs ayant participés à la partie ainsi que le nombre de voleurs capturés. Les gendarmes ne pouvant pas mourir, pour savoir combien de gendarmes ont participés à la partie il suffit d'accéder à la taille de la liste de gendarmes. On pouvait déduire du nombre de voleurs capturés toutes les autres statistiques intéressantes, notamment le nombre de voleurs enfuis : il suffit de prendre le nombre total de voleurs ayant participés et d'en retirer les voleurs capturés et ceux encore sur le terrain. Donnant une formule comme suit :

nbEnfuis=nbTotal-(nbCaptures+listeVoleursEnJeu.size())

C'est pour pouvoir faire ce calcul que lors de l'ajout d'un voleur nbTotal est incrémenté, mais lors du décès (et donc de la suppression) de celui-ci, nbTotal n'est pas décrémenté.

Enfin, lors de l'élaboration des IA, nous avons vite compris que nous devrions accéder aux données que possède la classe Jeu. C'est pourquoi nous avons placé les méthodes calculant les déplacement des personnages dans jeu, leurs permettant ainsi d'accéder à la liste des autres entités en jeu, et ainsi s'en servir pour calculer les déplacement. Ces méthodes prennent en paramètre le joueur dont elles calculent le coup.

1.2 Entités

On distingues les entités des personnages : après tout, peut-on vraiment dire qu'une sortie ou une cachette est un personnage ? Pas vraiment.

Il y a donc tout une famille de classes dédiée aux entités. Les entités à proprement parler sont liées par un point commun : leur présence sur le terrain. La position serait donc un attribut partagé pour toutes celles-ci, ainsi que leur hitbox.

Nous avons ajouté un ID pour différencier les entités. Pour éviter que deux entités de classes différentes aient le même ID, nous en avons fait une variable de classe d'Entité.

Il existe deux types d'entités : celles qui sont animées, et celles qui ne le sont pas. En somme, celles qui posséderont une intelligence et celles qui n'en auront pas. De là, nous avons donc créé deux classes filles à Entité : Joueur et NonJoueur.

NonJoueur regroupe toutes les entités qui n'ont pas d'intelligence artificielle. N'ayant aucun algorithme ou attribut pour les distinguer, nous avons créé une classe énumérée « Type » permettant de savoir quel est le NonJoueur traité : s'il s'agit d'un obstacle, d'une sortie, d'une cachette…

Joueur est bien plus complexe. C'est la classe regroupant tout ce qu'ont en commun les personnages et que n'ont pas les NonJoueur. Il s'agit donc de tout ce qui concerne l'IA et la gestion de déplacements.

Avant tout, il fallait sauvegarder la destination de chaque personnage. En effet, si nous ne le faisions pas, alors il y aurait un ordre de modification. Ça veut dire qu'un gendarme jouant avant un voleur aurait plus de chances de l'attraper, tandis qu'un voleur jouant avant un gendarme aurait plus de chances d'y échapper. Pour éviter ces cas de figure, nous avons fait en sorte que chaque joueur calcule son déplacement en fonction de la position actuelle des autres personnages. Il conserve alors en mémoire sa destination jusqu'à ce que tous les joueurs aient finis leurs calculs, et à ce moment là les changements sont tous appliqués. Une fois les changements appliqués, on vérifie si les hitbox des voleurs sont en contacte avec une autre entité : si c'est un gendarme, ils sont capturés. Si c'est une sortie, ils se sont enfuis.

Nous avions aussi besoin de savoir quel est le niveau d'intelligence du doueur. Pour cela, nous avons ajouté une classe énumérée « Choix_Algo » indiquant l'IA à utiliser pour calculer la direction empruntée par le Joueur.

Une fois la direction dans laquelle se dirige le Joueur obtenue, il ne reste qu'à y ajouter sa vitesse pour obtenir sa destination. C'est pourquoi nous avons aussi ajouté un attribut vitesse.

Bien que le cœur des IA soit dans Jeu.hh, Il y a quelques méthodes qui leurs sont utiles dans Joueur. Notamment les méthodes Fuir et Se_Rapprocher, permettant de fuir ou de se rapprocher d'un Joueur pris en paramètre. Pour se faire, on applique au joueur un vecteur correspondant à la direction opposée à laquelle se trouve le personnage qu'il veut fuir. Pour se rapprocher, il suffit d'aller à la direction opposée de celle de la fuite.

Les méthodes Gendarme_Plus_Proche et Sortie_Plus_Proche sont aussi utiles à tous les personnages, leurs permettant une meilleure visualisation des priorités (de quelle sortie se rapprocher en priorité, quel gendarme fuir en priorité,…). Pour cela, on prends en paramètre la liste de personnages dans laquelle on trouve celui le plus proche du Joueur.

Et enfin, les deux types de personnages existant : Les voleurs et les gendarmes. La grosse majorité de leurs fonctionnalités sont déjà présentes dans leur classe mère, Joueur. Ils ont cependant quelques différences:

Les gendarmes ont une méthode supplémentaire permettant de trouver le voleur le plus proche, qui a un fonctionnement similaire à gendarme_plus_proche, mais à usage unique des gendarmes cette fois. Non pas que les voleurs ne sont pas autorisés à connaître la position de leurs coéquipiers, mais c'est plutôt qu'ils n'en ont pas l'utilité.

Les voleurs, par contre, ont une méthode permettant d'éviter les murs. Cette méthode est particulière comparé aux autres méthodes utilisés lors du calcul de la direction, car elle a une priorité surpassant les autres. Ainsi, cette méthode prends en paramètre la direction que l'IA du voleur a calculé, et la modifie de manière à éviter de sortir du terrain. Pour cela, elle applique un vecteur de direction opposé au mur. L'ampleur de ce vecteur correspond au pourcentage de distance restant entre le joueur et le mur, dépendant de son champ de vision : si le voleur vient d'apercevoir le mur, il commence doucement à changer sa direction. Cependant, s'il l'a repéré il y a un moment et qu'il s'en rapproche encore, le vecteur d'opposition sera plus fort, jusqu'à arriver à un rejet total quand un voleur touche le mur.

Cependant, il y avait un soucis : si une sortie était collée à un mur, vu que la méthode permettant d'éviter un mur est appliquée avec une priorité absolue, elle empêchait les voleurs de s'enfuir, car ceux-ci ne pouvaient pas atteindre le mur où se

trouvait la sortie. C'est pourquoi nous avons ajouté une exception : un voleur peut s'approcher d'un mur uniquement s'il s'approche aussi d'une sortie.

Cette méthode permettant d'éviter les sorties de terrain n'est pas utilisée par les gendarmes, qui cherchent avant tout à poursuivre les voleurs. C'est les voleurs qui déterminent où vont ou non les gendarmes, et donc ces derniers n'ont pas besoin de vérification concernant la légalité de leur déplacement.

Les calculs de déplacement ainsi que l'utilisation de vecteurs de direction sont plus détaillés dans la section suivante, détaillant le fonctionnement des IA

2 Intelligences Artificielles

Il existe une partie commune à toutes les IA : les méthodes qu'elles utilisent. Notamment se_rapprocher, qui à été soumis à modification au milieu du projet. Au début, un joueur se rapprochait d'une entité en vérifiant s'il se situait en plus a l'Est ou plus à l'Ouest, auquel cas il attribuait l'une des trois valeurs -1,0 ou 1 à la valeur x de sa direction et il faisait de même avec le sud et nord pour sa valeur y.

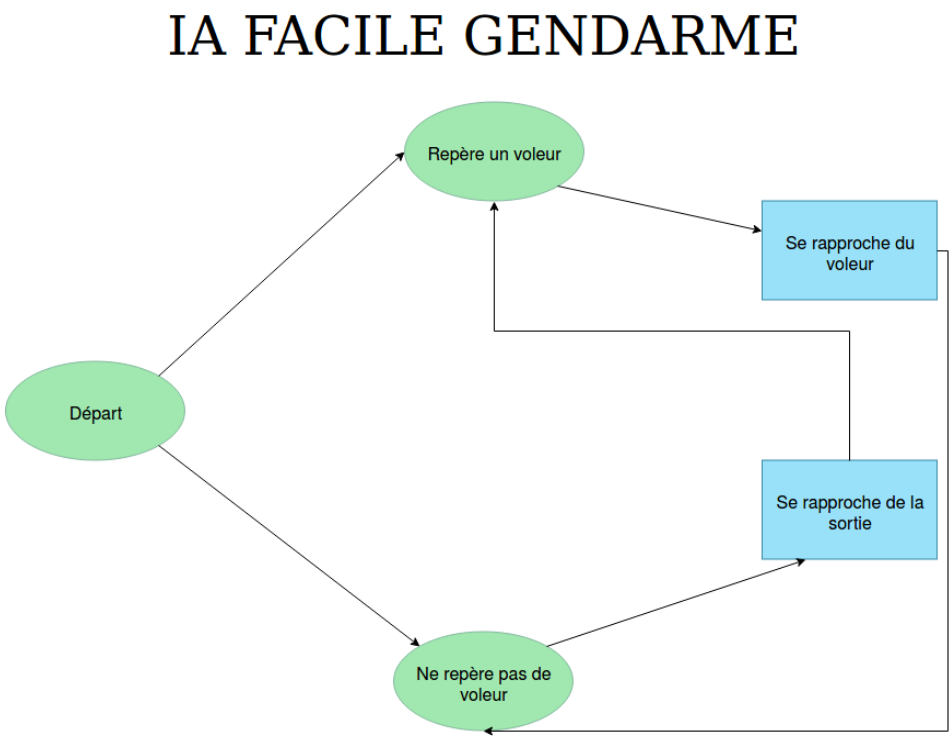
Le soucis de cette méthode, est qu'une entité se trouvant très légèrement au sud et une autre entité se trouvant extrêmement loin vers le sud sont traitées de la même façon. Ainsi, nous avons fait évoluer cette méthode pour avoir plutôt un vecteur partant du joueur, et dont le sommet est l'entité que le joueur veut atteindre. Les grandes valeurs obtenues ne sont pas importante, car on retourne seulement le ratio X/Y car il s'agit d'une direction (donc $|x|+|y|=1$). A partir de là, nous avons donc appliqué ce principe d'addition de vecteurs à l'ensemble des IA.

Ainsi, le calcul de la destination de l'IA a lieu en deux étapes : la première où des vecteurs sont additionnés pour obtenir une direction, et la deuxième où l'on modifie cette direction pour correspondre aux règles(pas de sortie de terrain,…) avant d'y ajouter la vitesse.

2.1 Gendarmes

Les gendarmes font dans la réaction. C'est à dire qu'ils prennent des décisions en fonction de celles prises par les voleurs. Ils sont donc un peu moins complexes que ces derniers, mais n'ont pas été délaissés pour autant.

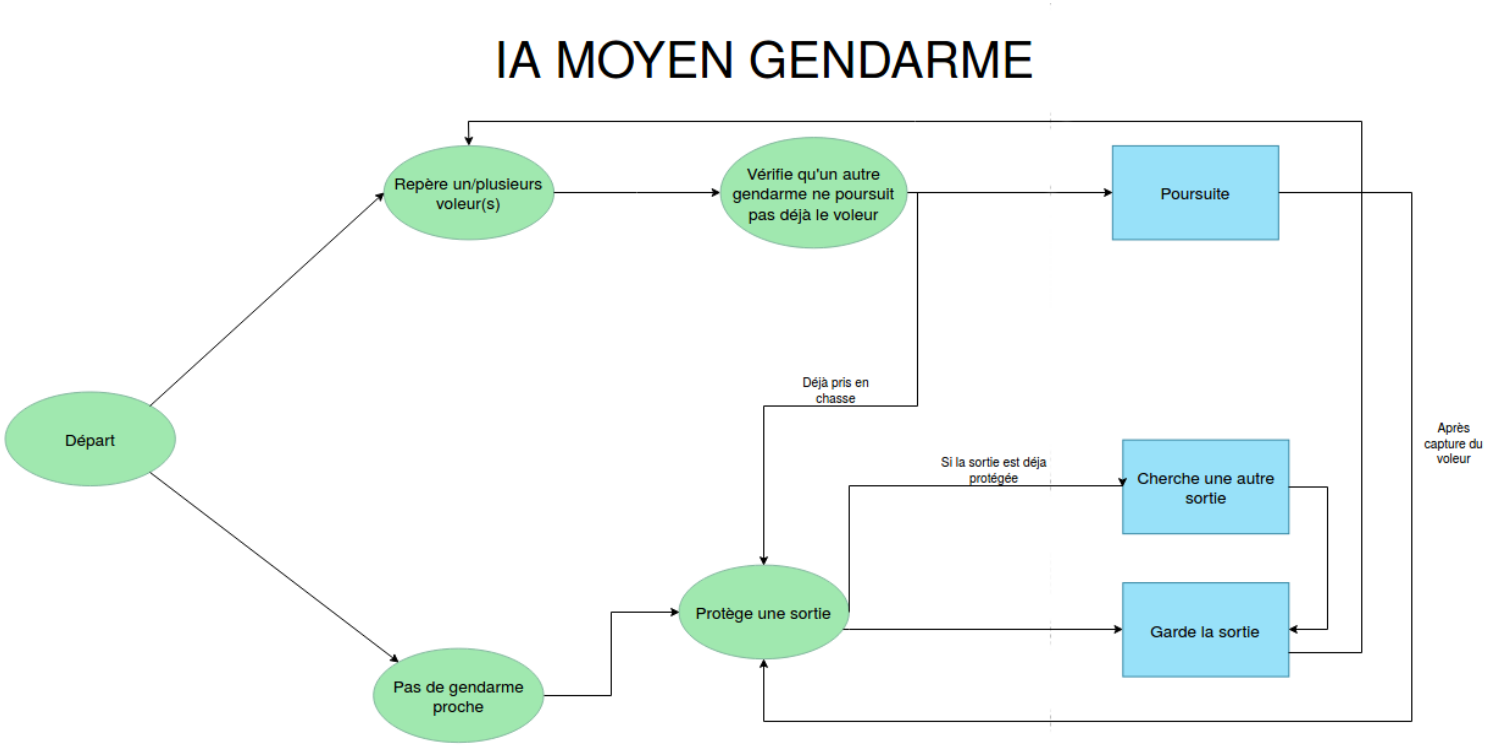
2.1.1 IA Facile



Arbre décisionnel de l'IA facile pour un gendarme

l'IA Facile des gendarmes n'a que deux comportements qui s'adaptent à la situation. S'ils ont repéré un voleur (un voleur est à une distance égale ou inférieure à la portée de vision du gendarme) alors ils vont se rapprocher de celui-ci, comme indiqué plus haut. Cependant, si aucun voleur n'est à portée, le gendarme va à l'endroit où il a le plus de chances de trouver un voleur : la sortie. Il se dirige donc vers la sortie la plus proche, et dés que celle-ci est dans sa portée de vue, il s'arrête et monte la garde. Faire ainsi permets aux voleurs de passer « en douce » s'ils font le tour pour entrer de l'autre côté de la sortie.

2.1.2 IA Moyen



Arbre décisionnel de l'IA moyen pour un gendarme

L'IA Moyen du gendarme a les mêmes objectifs que son IA Facile. La différence majeure est qu'elle prends en compte plus d'éléments pour prendre sa décision.

Avant de partir à la poursuite d'un voleur, le gendarme vérifie s'il n'a pas des collègues plus qualifiés pour le faire : il vérifie donc si un autre gendarme est plus proche que lui du voleur, et si celui-ci est déjà à sa poursuite. Dans ce cas, le gendarme ignore le voleur, et traite les autres qui sont dans son champ de vision. S'il n'y en a pas d'autres, il garde une sortie. Cela permet aux gendarmes d'agir plus efficacement, et de se relayer aux sorties le cas échéant.

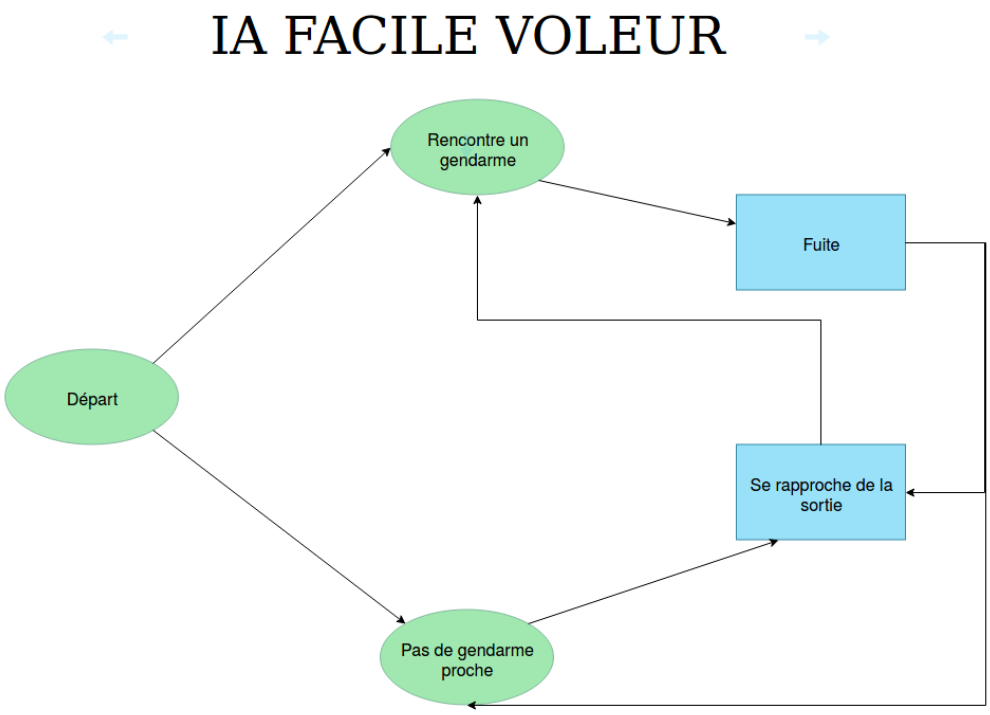
Concernant la garde de sortie, elle a aussi été optimisée. Désormais, le gendarme n'est pas à une distance de portée de vue de la sortie, mais il s'en approche bien plus, pour pouvoir vérifier qu'aucun voleur n'essaye de passer par derrière. Il est d'ailleurs suffisamment proche pour intercepter ces potentiels petits malins. Il vérifie aussi si un de ces collègues garde déjà la sortie. Si c'est le cas, il cherche s'il existe une autre sortie sans surveillance. Si c'est le cas, il se dirige donc vers la sortie concernée. Si ce n'est pas le cas, il rejoint son collègue à la garde de la sortie.

2.2 Voleurs

Les voleurs sont les joueurs menant le rythme du jeu. C'est eux qui sont l'élément « actif » là ou les Gendarmes sont dans la « réaction ». Ainsi, les seuls à pouvoir sortir du terrain étaient les voleurs. Ils sont donc les seuls soumis à cette vérification. Indépendamment de l'IA choisie, une fois que celle-ci rends une direction, cette direction est soumise à une modification en fonction de sa distance avec la bordure du terrain. Plus le voleur est proche d'une des bordures du terrain, plus sa direction est « tournée » vers l'intérieur du terrain. Cependant, si nous laissions la chose ainsi, il aurait été impossible pour les voleurs de gagner si une sortie était collée à la bordure du terrain. Ainsi, nous avons modifié la méthode pour que ce « rejet » appliqué par les bordures ne s'appliquent qu'aux voleurs qui s'éloignent de la sortie la plus proche.

Contrairement aux gendarmes, qui changeaient d'objectif en fonction de la situation (poursuivre le voleur s'ils en voient un, garder la sortie sinon), les voleurs prennent constamment en compte l'ensemble de leurs objectifs: (survivre, s'approcher de la sortie). Ainsi, nous avons ajouter des constantes d'importance pour retravailler facilement les priorités du voleur dans sa prise de décision. Chaque vecteur qui s'applique au voleur est ainsi renforcé en fonction de son importance : cela évite qu'un voleur soit immobile s'il essaye de s'approcher d'une sortie et qu'un gendarme se trouve au milieu. Il voudrait alors fuir le gendarme et approcher de la sortie, ce qui pousserait les vecteurs à s'annuler. Hors, avec les « poids » accordés aux différentes actions cela n'arrivera pas. Le voleur cherchera à fuir le gendarme en priorité, tout en cherchant la meilleure façon de fuir lui permettant de s'approcher de la sortie. Il cherchera donc ainsi à contourner le gendarme plutôt qu'à faire demi-tour.

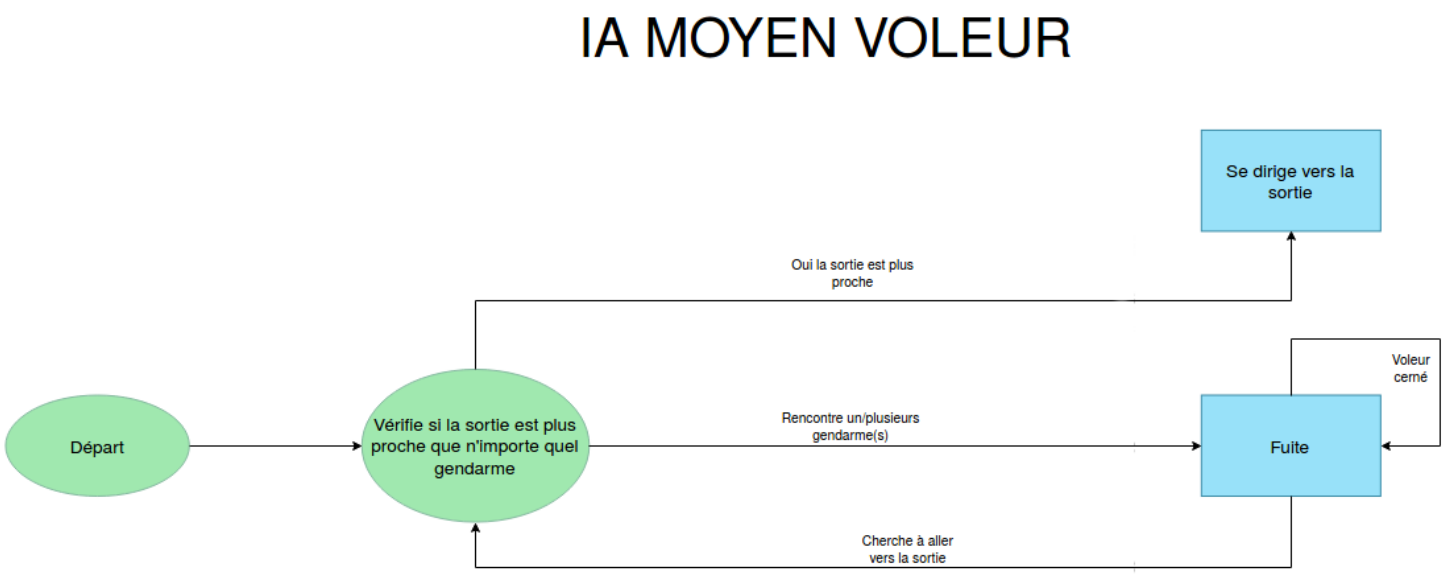
2.2.1 IA Facile



Arbre décisionnel de l'IA facile pour un voleur

L'IA facile du voleur le pousse seulement à survivre et approcher de la sortie. Il prends en compte un vecteur de fuite venant du gendarme le plus proche et un vecteur le poussant à s'approcher de la sortie la plus proche. Ainsi, le voleur ne prends qu'entre un et deux vecteurs en compte dans le calcul de sa direction. L'importance du vecteur de fuite du gendarme est trois fois plus importance que celle d'approche de la sortie.

2.2.2 IA Moyen



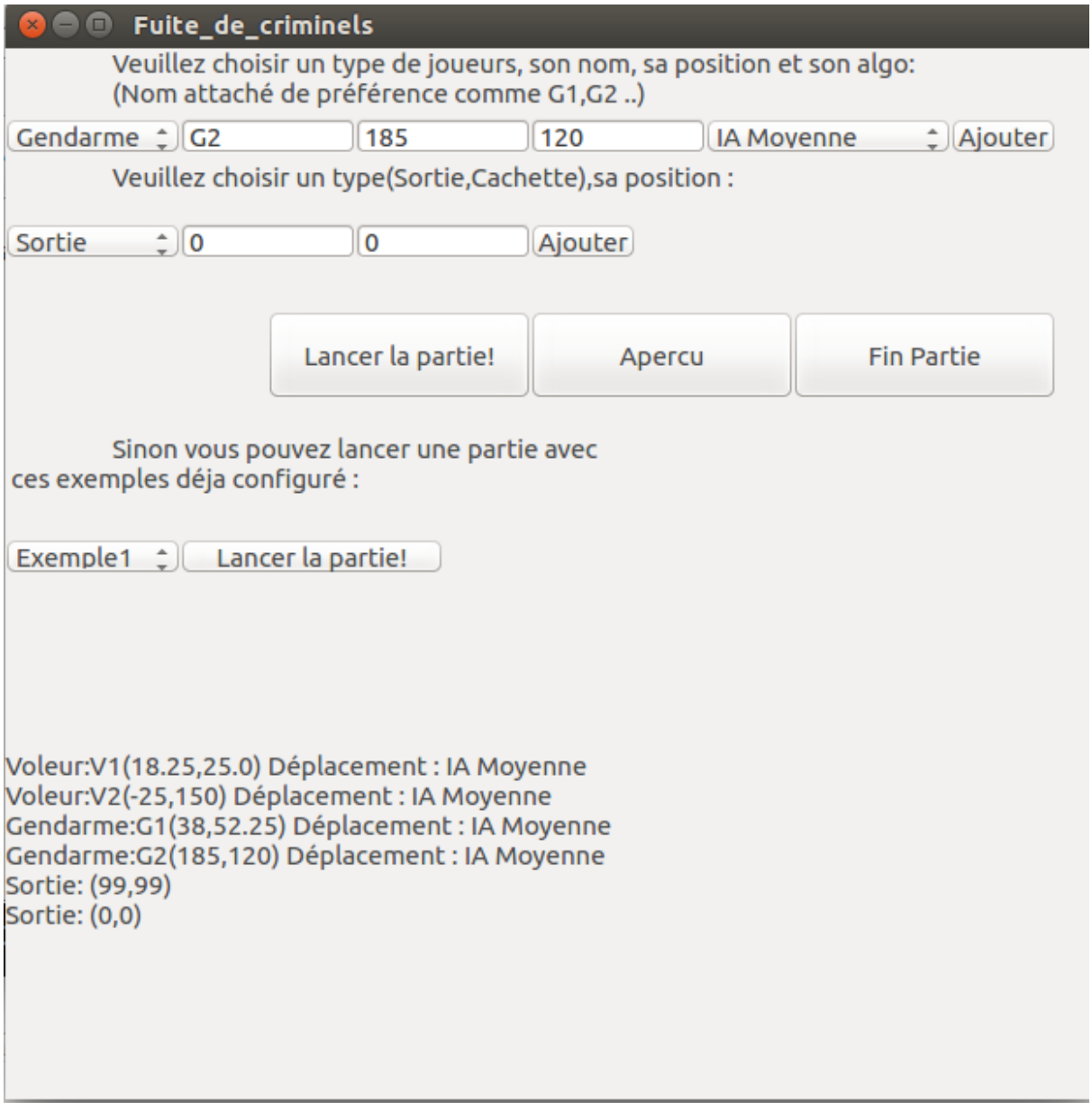
Arbre décisionnel de l'IA moyen pour un voleur

L'IA Moyen du voleur est cependant bien plus avancée. Le voleur fuit **tous** les gendarmes qu'il voit, avec une priorité doublée pour le gendarme le plus proche de lui. Les poids ont été ajustés pour éviter que le voleur ignore totalement la sortie quand il y a beaucoup de gendarmes à fuir. Cependant, faire ainsi pose un problème : le poids des vecteurs des gendarmes étant identiques et celui de la sortie très proche, le voleur pouvait se retrouver dans une situation où il resterait quasiment immobile s'il est cerné par des gendarmes et que le plus éloigné d'eux bloque la sortie. Nous avons donc instauré un système de mémoire tampon : on garde en mémoire le précédant vecteur d'influence traité. Ainsi, si le nouveau vecteur ajouté à la prise en compte annule le précédant, on garde le précédant et effectuons une rotation de 90° à celui-ci. Permettant ainsi de sortir des potentielles « prises en tenaille ».

Cette fois, le voleur vérifie aussi si un gendarme peut l'intercepter : s'il est plus proche d'une sortie qu'un gendarme ne l'est de lui et qu'il n'y a aucun gendarme entre la sortie et lui, il néglige l'influence de tous les gendarmes qu'il peut voir et se précipite sur la sortie.

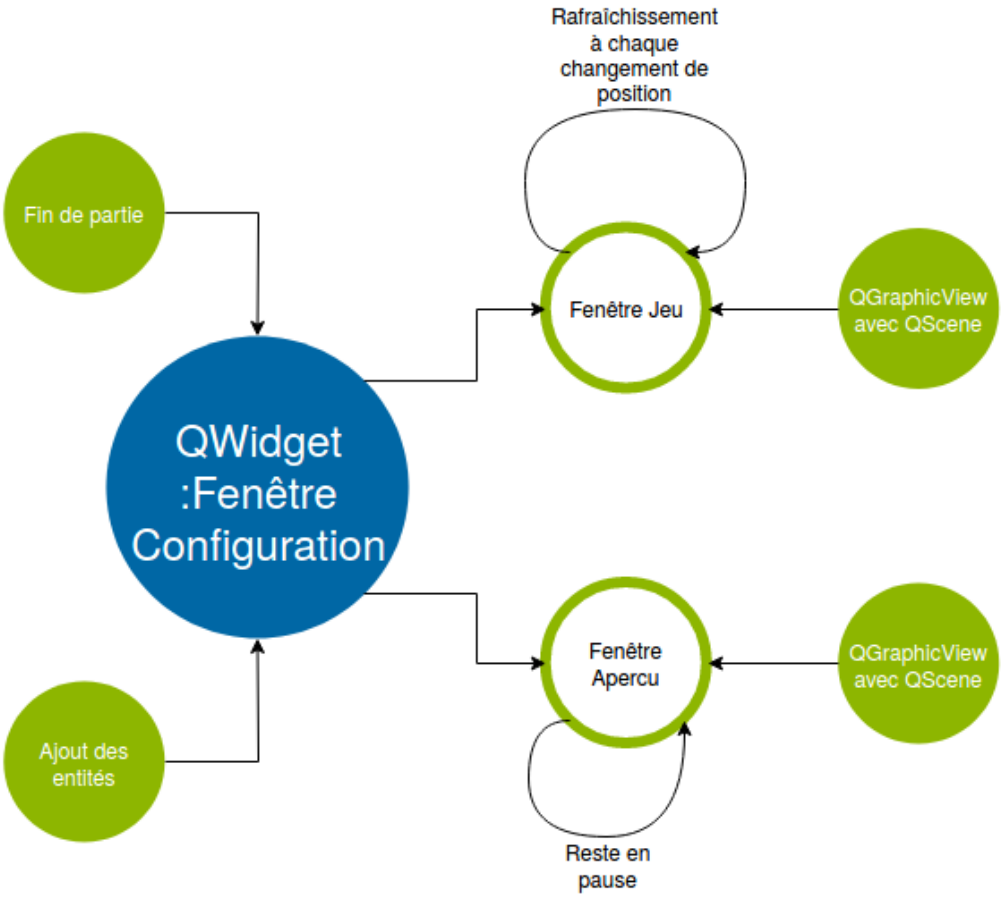
3 Partie GUI

Comme dit plus tôt aucun de nous deux n'avait vraiment touché à QT, sauf pour réaliser de simples calculatrices en cours. Nous avons dans un premier temps réalisé la fenêtre de configuration.



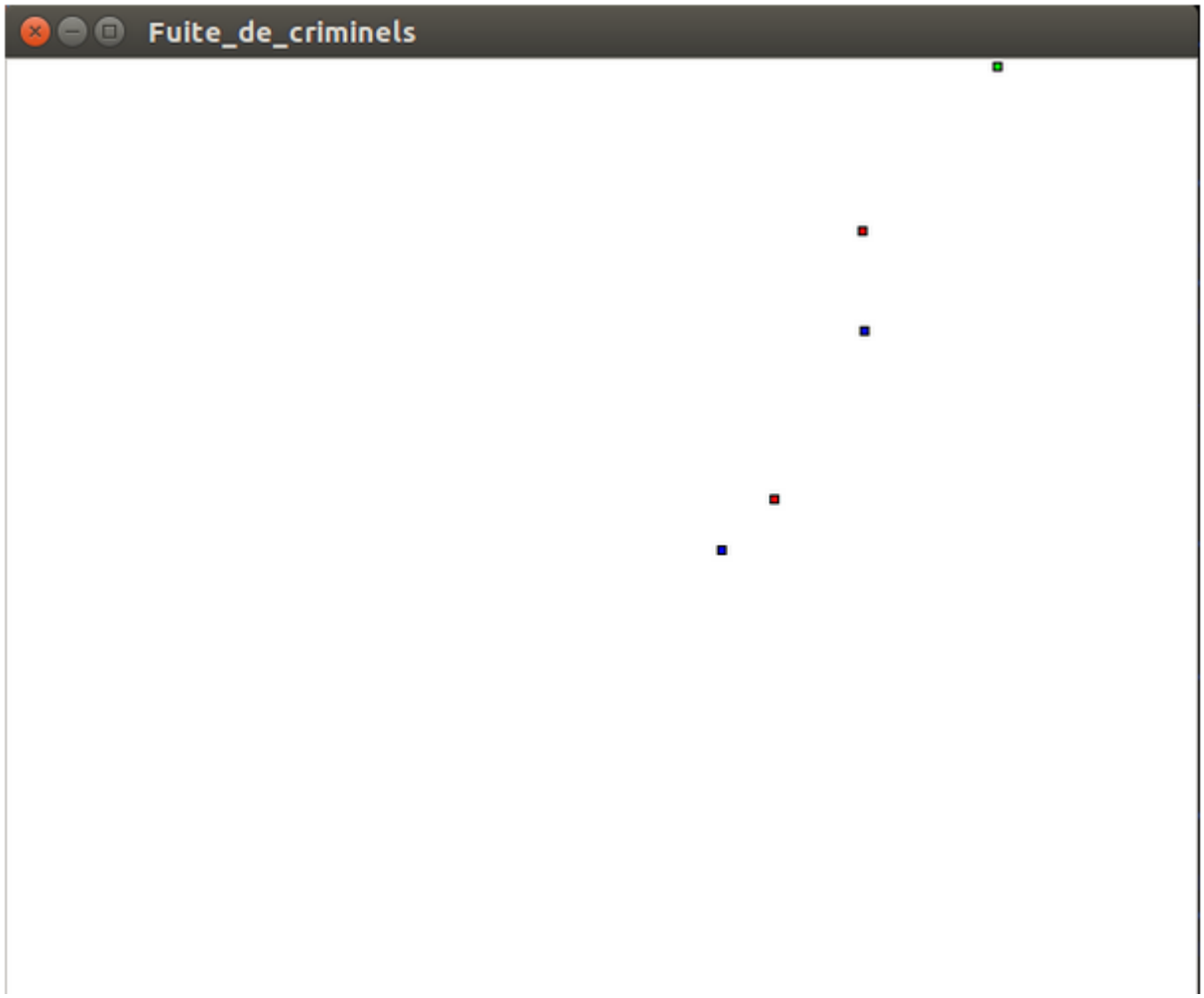
Fenêtre de configuration avec affichage des ajouts

Celle-ci permet d'ajouter manuellement des joueurs et de lancer la partie, ou bien d'utiliser des exemples déjà configurés qui n'attendent qu'à être lancé. La partie GUI permet par la suite de vérifier le déplacement de nos joueurs,le bon fonctionnement de la hitbox et enfin prendre en compte les murs.



L'utilisation du GUI

La fenêtre de configuration est relativement simple, elle n'a pas demandé énormément de temps. Il s'agit juste de saisir les données et d'initialiser la partie à l'aide celles-ci. Le second objectif fut l'ajout d'une fenêtre de jeu. Celle-ci affiche en direct le déroulement d'une partie au tour par tour et rafraîchit la position des QGraphicsRectItem utilisés pour représenter les joueurs.



Fenêtre de jeu. En vert -> sortie , en rouge -> voleur , en bleu -> gendarme

Nous avons choisit des couleurs très basique pour représenter nos entités: le gendarme est bleu, le voleur est rouge et la sortie est verte. Nous avons dû nous plonger dans la documentation de QT afin de comprendre comment on pourrait implémenter des formes géométriques, la fenêtre de jeu ainsi que la fenêtre d'aperçu. Elles sont ainsi composé d'un QGraphicView qui permet l'ajout d'un QScene.

La partie la plus délicate était l'affichage qui devait se rafraîchir au bouts de X millisecondes pour que les déplacements semblent naturel.

Dans un premier temps nous pensions pouvoir réaliser cet affichage à l'aide de la fonction sleep() présente en C++ permettant de mettre en pause le programme. Mais cela résultait en un écran noir sur la fenêtre de jeu puisque l'affichage repose sur l'exécution du processus.

Après quelques recherches, nous avons appris qu'il existe un QT event qui permet de créer une boucle que l'on inclus dans un QT Timer ce qui permet de faire une pause toutes les 100 millisecondes et donc donner l'impression que nos joueurs se déplacent de manière naturelle.

```
QEventLoop loop;
QTimer::singleShot(100, &loop, SLOT(quit()));
loop.exec();
Game->Afficher();
```

(Fichier Fenetre_Ajout.cc -> Jouer_Partie() , lignes 238-241)

La Fenêtre d'aperçu a comme fonction principale de visualiser l'état actuel du jeu durant sa configuration. Permettant ainsi à l'utilisateur de voir où sont situés les entités qu'il a placées avant de lancer la partie ou d'en ajouter d'autre. C'est une aide à la configuration de la partie. Une fois la partie lancée, le bouton permettant d'ouvrir l'aperçu ouvre la fenêtre de deu et non plus la fenêtre figée d'aperçu.

IV – Contenu supplémentaire

1 Génération de documentation automatique

On génère de la documentation automatiquement avec Doxygen. Celle-ci est sous différents formats : rtf, web... Pour éviter que les commentaires associés prennent trop de place, ceux-ci ont été placés dans des fichiers à part, eux même situés dans un dossier présent dans le code soure. Vous pouvez accéder à la documentation générée dans le dossier prévu à cet effet, *Documentation*.

2 Archives

Nous tenions aussi des archives pour pouvoir récupérer d'anciens tests ou d'anciennes méthodes. Cela permet de voir la progression le long du projet, et d'éviter de répéter d'anciennes erreurs. Les archives sont placées dans un dossier à part, tout comme la génération de documentation, pour éviter de surcharger le code source qui sera bien plus souvent consulté et retravaillé que les archives.

3 Tests

Des tests ont été réalisés tout au long de la conception du programme. Au début sous la forme d'un Main testant les fonctions et méthodes, puis ensuite avec des parties prédéfinies quand l'interface graphique fût disponible.

Nous avons tout de même gardé un suivi de l'exécution du programme dans le terminal pour vérifier que tout fonctionne correctement pendant le déroulement de la partie. Ça à permis, de nombreuses fois, de remonter à la source d'un comportement étrange de la part d'une IA ou bien d'une règle non respectée.

V-Déroulement du projet

1 Problèmes rencontrés

Nicolas : L'un des premiers problèmes rencontrés commun à nous deux à était la distanciation provoqué par la crise sanitaire, en effet pouvoir travailler dans les locaux de la faculté aurait été un plus pour nous deux, une interaction en direct nous aurait permis d'avancer plus vite et de travailler en même temps, en effet vue la situation nous n'avions pas le même emploi du temps, entre problèmes familiaux et problèmes de collocations nous avons perdu du temps. Mon second problème fut que mon ordinateur se mette à ne plus fonctionner plus d'une semaine avant la fin du projet.. Rappelons quand même que je venais de l'acheter il y a 4 mois.

J'ai perdu une bonne semaine et j'ai été contraint d'en racheter un autre pour revoir le code et rédiger le rapport en attendant son retour du SAV. J'en suis encore désolé et je remercie Baptiste de n'avoir rien lâché de son côté pendant mon "absence".

Baptiste: Comme Nicolas l'a précisé, beaucoup de soucis ont été engendrés par le confinement, notamment concernant la communication et l'organisation. J'ai eu quelques soucis de famille de mon côté, mais cela ne m'a fait perdre que quelques jours. Cependant, ces différentes pertes de temps et le fait de ne pas avoir la durée prévue initialement dans le programme nous à empêchés de terminer toutes les IA et autres fonctionnalités que vous voulions implémenter, tels que les cachettes ou les idées du point qui suit.

2 Idées supplémentaires

Nous avions plusieurs idées qui sont apparus en cours de projet, ne serait-ce dans un premier temps au niveau de l'implémentation de nos joueurs en travaillant avec des angles. Ajouter des cachettes afin de permettre aux voleurs de se cacher s'ils étaient pris en chasse par un gendarme, ainsi que des obstacles où nous aurions pu implémenter un algorithme de pathfinding (A* ou bien Dijkstra) afin de les éviter.

Au niveau de l'interface graphique, nous aurions pu ajouter une fonctionnalité d'ajout sur un terrain vierge à l'aide de la souris plutôt que d'ajouter manuellement les entités. Un bouton supprimer après l'ajout d'un joueur si nous avions fait une erreur, alors qu'ici un clic

sur "Fin de partie" réinitialise tous les ajouts effectués. Un affichage du score en direct dans la fenêtre de jeu aurait pu être un plus.

Au niveau des IA, on aurait voulu implémenter une IA difficile ainsi qu'une IA élite pour chaque joueur. Nous aurions voulu ainsi avoir une IA Facile qui soit la base, une IA Moyenne qui ait les mêmes comportements que l'IA Facile mais avec une prise de décision améliorée, une IA Difficile ayant de nouveaux comportements bien plus complexes, et une IA Elite ayant les mêmes comportements que l'IA difficile mais avec une prise de décision optimisée.

Une idée de "communication" ou "talkie-walkie" entre les gendarmes pouvant provoquer des déplacements de groupe ou des alertes en cas de repérage de voleur(s). Cette fonctionnalité à été remplacée par le fait que les gendarmes aient accès aux informations des uns des autres. Plus de détails sur cette fonctionnalité dans le dossier « temporaire » du code.

Nous avions aussi envisagé la possibilité d'un champ de vision directionnel plutôt qu'un cercle de détection autour des Joueurs. Nous avions d'autres idées (sprint,prison,obstacles…) qui ont été listées au fur et à mesure du projet dans le fichier idees.txt disponible avec le code fourni.

VI-Bibliographie, documentation

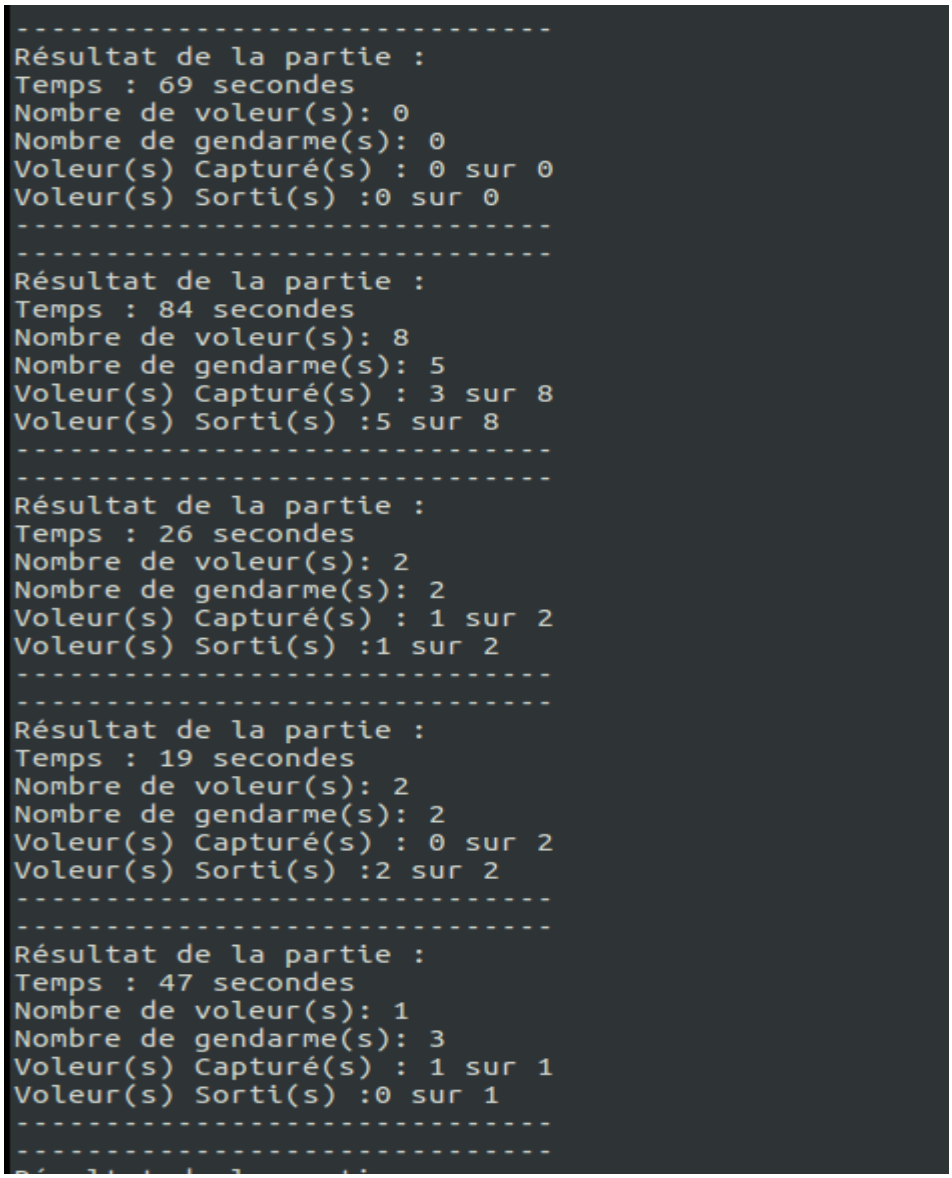
https://doc.qt.io/ -> Interface graphique QT

https://www.cplusplus.com/ -> C++

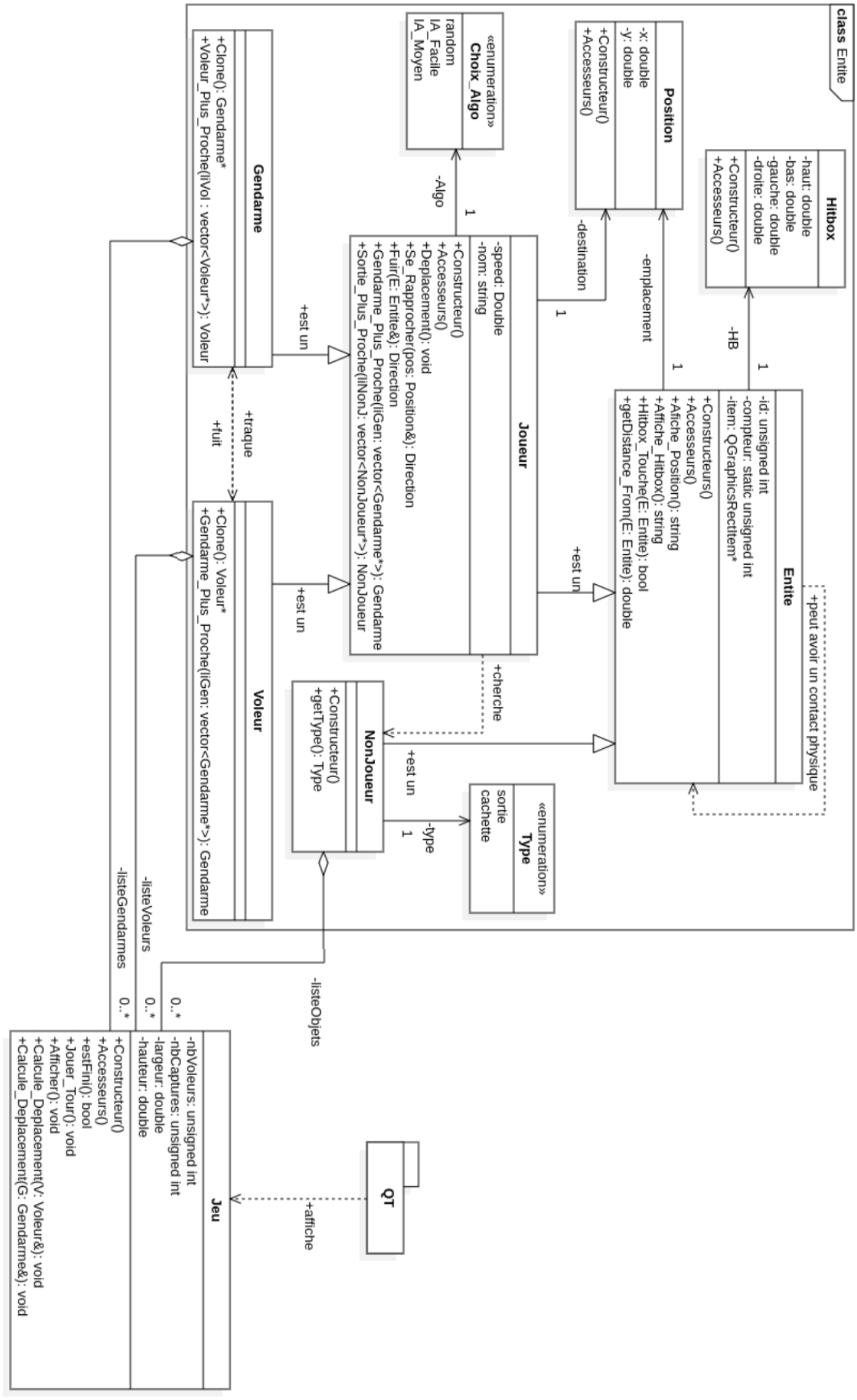
https://online.visual-paradigm.com -> Arbre de décision, schéma

Et les différents cours que nous avons eu au cours de l'année.

ANNEXES



Exemple de résultat présent dans Resultat.txt II - 2.3


class Entite
Hitbox
-haut: double
-bas: double
-gauche: double
-droite: double
+Constructeur()
+Accesseurs()
Position
-x: double
-y: double
+Constructeur()
+Accesseurs()
«enumeration»
Choix_Algo
random
IA_Facile
IA_Moyen
Entite
-id: unsigned int
-compteur: static unsigned int
-item: QGraphicsRectItem*
+Constructeurs()
+Accesseurs()
+Affiche_Position(): string
+Affiche_Hitbox(): string
+Hitbox_Touche(E: Entite): bool
+getDistance_From(E: Entite): double
+peut avoir un contact physique
Joueur
-speed: Double
-nom: string
+Constructeur()
+Accesseurs()
+Deplacement(): void
+Se_Rapprocher(pos: Position&): Direction
+Fuir(E: Entite&): Direction
+Gendarme_Plus_Proche(liGen: vector<Gendarme*>): Gendarme
+Sortie_Plus_Proche(liNonJ: vector<NonJoueur*>): NonJoueur
Gendarme
+Clone(): Gendarme*
+Voleur_Plus_Proche(liVol : vector<Voleur*>): Voleur
Voleur
+Clone(): Voleur*
+Gendarme_Plus_Proche(liGen: vector<Gendarme*>): Gendarme
NonJoueur
+Constructeur()
+getType(): Type
«enumeration»
Type
sortie
cachette
-HB
-emplacement
-destination
-Algo
+est un
+cherche
+traque
+fuit
-type
-listeObjets
-listeVoleurs
-listeGendarmes
0..*
1
Jeu
-nbVoleurs: unsigned int
-nbCaptures: unsigned int
-largeur: double
-hauteur: double
+Constructeur()
+Accesseurs()
+estFini(): bool
+Jouer_Tour(): void
+Afficher(): void
+Calcule_Deplacement(V: Voleur&): void
+Calcule_Deplacement(G: Gendarme&): void
QT
+affiche