Analyse du Démineur

Clément - **** L3

Le démineur est un **jeu de réflexion** dont le but est de localiser des mines cachées dans un champ virtuel avec pour seule indication **le nombre de mines dans les zones adjacentes**. Le jeu fait appel à la **logique** du joueur, mais aussi à sa **chance**, en effet le Démineur comporte des aspects de hasard, notamment pour les premiers coups du joueur. Le jeu est **chronométré**, ce qui permet de comparer les scores.

Le champ de mines est représenté par une **grille en deux dimensions**, avec des cellules (aussi appelées cases), tout comme les automates du projet précèdent. Chaque cellule a **8 voisines** (droite, gauche, haut, bas et diagonales)
Chacune des cases de la grille a une chance de cacher **une mine**. Le but du jeu est de découvrir toutes les cases vides sans faire exploser les mines, c'est-à-dire sans découvrir une seule des cases qui les dissimulent.

Il faut pour cela, créer une **grille en 2D**, contenant les cellules, chaque cellule pourra être une mine, ou non. Graphiquement parlant, les cellules seront toutes pareilles, notamment au début de la partie (à l'initialisation). On ne pourra pas les différencier. La réside tout l'intérêt du jeu, qui est de supposer et trouver l'emplacement des mines. La position des mines sur le plateau de jeu sera révélée à la fin de la partie (qu'elle soit gagnée, ou perdue).

Lorsque le joueur clic sur une case libre comportant **au moins une mine** dans l'une de ses cases avoisinantes (ses voisines), **un chiffre apparaît**, indiquant ce nombre de mines. Si en revanche toutes les cases adjacentes sont vides, une case vide est affichée et **la même opération est répétée** sur ces cases, et ce jusqu'à ce que la zone vide soit entièrement délimitée par des chiffres. En comparant les différentes informations récoltées, le joueur peut ainsi progresser dans le déminage du terrain. S'il se trompe et clic sur une mine, **il a perdu**.

Deux fonctions seront nécessaires pour l'initialisation graphique, une pour **la création de la grille**, une pour **la création des cellules** dans cette grille. Au début de la partie, toutes les cellules seront dites « **Inconnues** ». Nous ne savons rien de ce qui se cache derrière. C'est pourquoi on parle de **hasard** pour les premières actions du joueur, on a une chance de tomber directement sur une mine, et perdre la partie, ou encore de tomber sur une cellule avec des mines dans son voisinage, ne dévoilant que cette cellule-là.

Fonction -> CreationGrille
In -> NombreDeLigne, NombreDeColonne
Out -> Null

La fonction prendra en paramètre deux valeurs, le nombre de ligne et le nombre de colonne que la fonction devra dessiner. Elle ne retourne aucune valeur. La création de la grille se fera avec des dimensions fixes selon la difficulté du jeu, il ne sera donc pas forcement nécessaire de les passer en paramètre.

Fonction -> InitialisationDesCellules
In -> NombreDeLigne, NombreDeColonne
Out -> Null

La fonction prendra en paramètre deux valeurs, le nombre de ligne et le nombre de colonne afin de dessiner les cellules. Elle ne retourne aucune valeur.

Comme dit, les cellules pourront être **piégées**, ou être **vides**, le nombre de mine à poser devra être déterminé avant le début de la partie. On peut déjà imaginer **plusieurs niveaux de difficulté**, qui détermineront le nombre de mines dissimulées sur le plateau, plateau qui pourra être plus ou moins grand, encore une fois selon la difficulté **choisie par le joueur**.

Les difficultés seraient au nombre de **quatre** (facile, moyen, difficile, impossible). Plus la partie sera compliquée, plus le nombre de mines à placer et la taille du plateau seront élevés.

Deux fonctions seront nécessaires, une pour changer la difficulté du jeu, et une qui changera les paramètres de la partie en fonction de cette difficulté.

Fonction -> ChangementDifficulter
In -> Null
Out -> Null

La fonction ne prend aucun paramètre ni ne retourne aucune valeur. La fonction se contentera d'incrémenter le niveau de difficulté jusqu'à revenir au premier.

Fonction -> DefinirOptionDuJeu
In -> DifficulterDuJeu
Out -> Options

La fonction prendra en paramètre la difficulté du jeu. Elle retournera une liste des options du jeu (nombre de mine à poser, taille de la grille et nombre de cellules). Elle changera les options de la partie en fonction de la difficulté actuelle / passée en paramètre.

Pour pouvoir placer des mines aléatoirement dans notre jeu, il faut avoir accès à un **générateur de hasard**. On lui demanderait un nombre compris entre 1 et n, ce sera le numéro de la ligne, et un second nombre compris entre 1 et m, ce sera le numéro de colonne. Nous aurons ainsi les coordonnées d'une cellule. Ensuite on regarderait cette case, si elle ne comporte pas déjà une mine, on y place une mine, sinon on recommence. On effectue ce schéma jusqu'à ce qu'on ait placé x mines sur le plateau de jeu.

Malheureusement, après réflexion, cet algorithme **est gourmand en ressource** lorsqu'il reste peut de cases vides, la probabilité de tomber sur une case déjà occupée par une mine étant de plus en plus grande.

Afin de placer les mines de façon **aléatoire** et d'une autre manière que précédemment, nous pourrions faire une liste des cellules où l'on peut placer une mine. Nous pourrons ainsi décider les cellules où placer des mines, mais aussi éliminer les cellules où l'on ne veut pas en placer (les coins de la grille par exemple).

Nous tirerons au hasard une cellules présente dans la liste des cellules, une cellule qui n'est donc **pas encore piégée**, après avoir posé une mine sur celle-ci, **nous retirerons la cellule de la liste**. Ainsi nous n'aurons aucune chance de tomber sur une cellule avec une mine déjà présente. C'est **cette méthode** que nous retiendrons pour notre jeu.

Fonction -> PoseDesMines
In -> NombreDeMineAPoser
Out -> Null

La fonction prendra en paramètre le nombre de mine à placer sur le plateau. Elle ne retourne aucune valeur. Elle placera aléatoirement des mines selon une méthode décrite précédemment.

Comme dit précédemment, les cellules avec des mines dans leur **voisinage ont un chiffre** avec le nombre de ces mines sur celle-ci. Pour déterminer ce chiffre (le nombre de mines dans le voisinage de chaque cellule), je vois **2 solutions possibles**.

La première approche, nous parcourons **toutes les cellules**, puis analysons chacune de leurs voisines afin de compter le nombre de voisines ayant une mine, s'il n'y a aucune mine, on ne place pas de chiffre sur la cellule. Cette méthode peut s'avérer **longue**, et va analyser des cellules **inutilement**, notamment les cellules avec aucunes cellules piégées.

La deuxième approche, plus compliquer à mettre en place mais plus rentable au niveau des performances, à chaque mine posée, nous incrémentons de 1 le nombre de bombe dans le voisinage des voisines. Cela sera plus **optimiser** du fait que nous n'avons pas besoin de parcourir toutes les cellules. On réalise cette opération **pendant le placement des mines aléatoires** et plus à la fin.

Nous choisirons la deuxième approche.

L'affichage des chiffres sur les cellules, encore une fois peut se passer de deux manières au premier abord. Nous pouvons créer tous les textes en premier lieu, après la pose des mines, puis les afficher en allant après la découverte des cellules. Nous pourrions au contraire les créer, et donc les afficher en même temps que la découverte des cellules, ce que nous ferons ici, pour éviter les temps de latences lors de la création de beaucoup d'objets en même temps.

Pour l'interaction Joueur – Application, il nous faut des « commandes » **intuitives**, pour la découverte d'une cellule, nous choisirons **le clic gauche de la souris**.

Fonction -> ClicGaucheSouris
In -> CoordXDeLaCellule, CoordYDeLaCellule
Out -> Null

La fonction ne retournera aucune valeur, elle prendra en paramètre les coordonnées de la cellule à découvrir par la suite.

La découverte d'une cellule peut se faire si cette cellule **n'a pas de drapeau** et si elle n'est **pas déjà découverte**. Ce sont les premières conditions. Nous créerons donc une fonction découverte d'une cellule prenant en paramètre les coordonnées de cette cellule à découvrir.

Nous savons que la partie se termine et gagnée lorsque le joueur **découvre toutes les cellules** non piégées du jeu, ainsi dans la fonction de découverte d'une cellule, il faudra vérifier qu'il y a encore des cellules non découverte sur le plateau (n'étant pas des mines) pour continuer, sinon on termine la partie (le joueur gagne).
Lors de la découverte de la cellule, il faut également vérifier qu'il **ne s'agit pas d'une mine**, ici pareil, **on termine la partie** (le joueur perd)

Lorsque l'on découvre une cellule, comme dit précédemment, si elle n'a aucune voisines piège, on découvre toute celles autour de manière à découvrir toutes les cellules jusqu'à n'avoir que des cellules avec un chiffre aux frontières des zones découvertes.

Pour faire cela, nous aurons besoin d'une condition vérifiant que la cellule a des mines dans son voisinage, au cas contraire on rappellera la fonction de découverte avec pour paramètre les coordonnées des dites cellules voisines.

Fonction -> DecouvrirLaCellule
In -> CoordXDeLaCellule, CoordYDeLaCellule
Out -> Null

La fonction ne retournera aucune valeur, elle prendra en paramètre les coordonnées de la cellule à découvrir.

Le clic droit, quant à lui, servira à poser un drapeau, qui **signalera** une potentielle cellule piégée, le joueur ne pourra plus cliquer sur cette cellule signalée. S'il veut la découvrir malgré cela il devra enlever le drapeau.
Le clic droit servira également à poser un point d'interrogation, lorsque le joueur a un **doute** sur une cellule, le joueur pourra la découvrir sans avoir besoin de l'enlever.

Le clic droit pourra également faire revenir à l'état **initial** une cellule (enlever point d'interrogation).

Pour résumer, un clic droit sur une cellule inconnue, posera un drapeau sur cette cellule, un clic droit sur une cellule avec un drapeau, enlèvera le drapeau et placera un point d'interrogation, un clic droit sur un point d'interrogation, enlèvera se point d'interrogation et la remettra dans son état initial.

Fonction -> ClicDroitSurLaCellule
In -> CoordXDeLaCellule, CoordYDeLaCellule
Out -> Null

La fonction ne retournera aucune valeur, elle prendra en paramètre les coordonnées de la cellule avec qui interagir par la suite.

Quatre fonctions seront nécessaires pour l'interaction avec les cellules. Une sera pour la pose d'un drapeau, une autre pour la pose d'un point d'interrogation, une autre pour réinitialiser la cellule et la remettre à son état initial, la dernière pour **administrer** le tout et déterminer que faire en fonction de l'état de la cellule.

Fonction -> PoseDrapeauSurCellule
In -> CoordXDeLaCellule, CoordYDeLaCellule
Out -> Null

La fonction ne retournera aucune valeur, elle prendra en paramètre les coordonnées de la cellule sur laquelle on posera un drapeau.

Fonction -> PosePointInterogationSurCellule
In -> CoordXDeLaCellule, CoordYDeLaCellule
Out -> Null

La fonction ne retournera aucune valeur, elle prendra en paramètre les coordonnées de la cellule sur laquelle nous poserons un point d'interrogation.

Fonction -> ReinitialiserCelluleInconnue
In -> CoordXDeLaCellule, CoordYDeLaCellule
Out -> Null

La fonction ne retournera aucune valeur, elle prendra en paramètre les coordonnées de la cellule à réinitialiser.

Fonction -> ChangerEtatDeLaCellule
In -> CoordXDeLaCellule, CoordYDeLaCellule
Out -> Null

La fonction ne retournera aucune valeur, elle prendra en paramètre les coordonnées de la cellule qu'il faut changer, selon son état actuel (si cellule inconnue, on pose un drapeau, si on a déjà un drapeau de poser, on pose un point d'interrogation, si un point d'interrogation est déjà posé, on réinitialise la cellule).

Pour terminer, il nous faudra une fonction qui va **gérer la fin de partie**, informant le joueur du résultat de la partie, ainsi que son temps de jeu. A savoir que le temps de jeu serait en seconde et commencerait à partir de la première interaction du joueur, et non à partir de la création de la partie.

Fonction -> FinDePartie
In -> TempsDeJeu, JoueurAGagne
Out -> null

La fonction ne retournera aucune valeur, elle prendra en paramètre les résultats de la partie (le temps de jeu et si le joueur a gagné ou perdu) puis affichera le tout au joueur.

Mais, également une fonction pour en créer une, on peut imaginer qu'à chaque création d'une nouvelle partie nous remettons tout à zéro, nous redessinons la grille ainsi que les cellules.

Fonction -> CreationNouvellePartie
In -> DifficulterDuJeu
Out -> Null

La fonction ne retournera aucune valeur, elle prendra en paramètre les options de la partie (la difficulté). La grille et les cellules se feront en fonction de la difficulté passer en paramètre.

Pour réaliser notre programme nous utiliserons **Python**, pour sa simplicité et sa librairie **Tkinter**, mais aussi car nous avons déjà réalisé un projet dans ce même langage (les automates cellulaires) avec du code et des fonctions que nous pourrons **adapter** à notre cas. Je pense notamment à la création de la grille et à l'analyse des cellules voisine.

Analyse de l'Intelligence artificielle du Démineur

L'intelligence artificielle est « l'ensemble des théories et des techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence » - Wikipédia.

Le jeu du démineur étant **terminé et fonctionnel.** Passons maintenant à la création de l'intelligence artificielle (que nous appellerons IA ou encore AI). Elle jouera et gagnera des parties de notre jeu. Elle sera codée en **Python**, tout comme notre démineur précédemment.

Mon but est qu'elle soit directement **incluse** dans celui-ci, avec la possibilité de l'activer et de la désactiver **pendant la partie**. Le joueur pourra jouer en même temps qu'elle. Cela pourra amener le joueur à « **coopérer** » avec l'IA, et à amener l'IA à **débloquer** le joueur lorsqu'il en a besoin.

L'IA aura une **confiance aveugle** dans les actions du joueur, c'est-à-dire que si le joueur pose un drapeau à un endroit, l'IA considèrera que c'est bon et qu'il y a une mine en dessous. Il sera également possible de régler le temps entre les actions de l'IA (sa **vitesse** de jeu).
Je vais baser mon IA sur les mouvements, la réflexion que l'on peut avoir lors d'une partie de jeu afin de gagner en un minimum de temps.

Pour cela il va falloir la faire analyser **toutes les cellules,** déterminer lesquelles sont de **potentielles mines** (les cellules où il faudra placer un drapeau) et les cellules qu'il faudra **découvrir**.
Pour rappel, on perd une partie lorsque l'on découvre une mine. On gagne lorsque toutes les cellules sont découvertes (sauf celles piégées). Les mines sont dissimulées sur le plateau de jeu de manière aléatoire.

Dans le jeu du démineur, les premiers coups (nos premiers clics sur des cellule choisis sur le plateau, de façon aléatoire) sont **décisifs** pour la suite de la partie. Malgré cela, ils dépendent exclusivement de la **chance** du joueur, ou de l'IA dans notre cas.

Notre premier coup peut nous faire perdre instantanément dans le cas où pour notre première cellule découverte, nous tombons sur une mine. Cela arrive assez souvent, notamment dans les parties avec une difficulté impliquant de nombreuses mines sur le plateau.
Dans le cas où nous découvrons une cellule, et non une mine, il pourrait s'agir d'une case contenant des mines dans son voisinage, ne dévoilant ainsi que cette case, avec pour seules informations, le nombre de ses 8 voisines piégées (le nombre s'affichant sur cette dite case).

Dans ce cas il est possible d'opter pour plusieurs stratégies pouvant s'avérer plus ou moins efficaces. On pourrait par exemple, cliquer sur une autre case de manière aléatoire sur le plateau en reprenant les risques du mouvement précèdent. Nous pourrions ainsi tomber sur un bloc de cellules ne contenant aucunes cellules piégées dans ses voisines, ce qui découvrirait plusieurs cellules pour nous permettre d'avancer dans le jeu sans compter exclusivement sur la chance mais de manière réfléchie.
Nous pourrions, au contraire, décider de tenter notre chance avec les voisines de la cellule que l'on vient de découvrir. En fonction de son nombre de mine dans son voisinage, on aura plus ou moins de chance de tomber sur l'une d'entre elles. Ainsi nous pourrions essayer de découvrir un maximum de cellules dans les alentours, en misant sur les cellules avec le moins de cellules piégés dans leur voisinage jusqu'à ne plus avoir besoin de mélanger chance et stratégie, pour agir de manière réfléchie.

Malheureusement les deux stratégies ont leur limite et leurs chances de réussir, par exemple dans le premier exemple nous pourrions tomber plusieurs fois de suite sur des cellules avec des mines aux alentours, augmenter la chance de tomber sur une mine et finir par tomber sur une d'elle, et perdre la partie.

La deuxième méthode aura beaucoup de chance d'échouer si la cellule découverte a un grand nombre de ses voisines avec des mines, et même avec un faible nombre, la probabilité de tomber sur une mine est forte (Nombre de mine dans le voisinage / 8).

Pour notre IA, nous découvrirons des cellules de manière aléatoire tant que nous ne pouvons agir sur des cellules de manière stratégique. C'est-à-dire que si nous n'avons aucune cellule inconnue, où l'on peut poser un drapeau, où l'on peut découvrir avec toutes les chances que ce soit bon, nous procèderons de manière aléatoire. La stratégie peut encore changer en fonction de l'avancement du projet.

Dans le but d'analyser les cellules du jeu, d'interagir avec elles, je peux imaginer plusieurs moyens. Nous pourrions parcourir les cellules déjà découvertes afin d'agir sur leurs voisines directement, de manière à tester les cellules avec leurs voisines découvertes et inconnue.

Malheureusement je peux déjà voir les limites de cette méthode. En effet, il y a des situations en jeu où l'on ne peut se baser seulement sur les voisines d'une cellule pour déterminer le futur de la voisine, il faudrait les autres voisines de cette voisine.

Au contraire, nous pourrions parcourir les cellules NON découvertes, de façon à se mettre à la place de la cellule inconnue, et voir si ses voisines impliqueraient un changement chez elle, serait-elle entourée de cellule indiquant qu'une seule mine est placer dans leurs alentours ? cela impliquerait que cette même cellule inconnue deviendrait une cellule potentiellement piéger. Cela implique alors d'avoir une liste des cellules non découvertes. Encore une foi, il nous faudrait les voisines de ces cellules inconnues.

Apres réflexion, nous procéderons comme cela : Nous parcourrons la liste des cellules inconnues, de manière a analysé seulement les cellules utiles. Tant que nous n'aurons pas trouver de cellule inconnue intéressantes où interagir, nous continuerons de chercher dans la liste. Comme dit précédemment, si nous ne trouvons aucunes coordonnées intéressantes dans la liste, nous choisirons au hasard parmi les cellules inconnues.

Afin de déterminer si une cellule inconnue est une bonne cellule avec laquelle interagir (de manière stratégique, en suivant les règles du démineur), il faudra analyser ses voisines, et les voisines de ses voisines, il faut comparer le nombre de cellule voisine et le nombre de cellule découverte. Il faut également comparer le nombre de drapeau dans le voisinage et le nombre de mine dans celui-ci. Dans le cas où les valeurs seraient égales, la cellule inconnue pourra être découverte, dans le cas contraire la cellule inconnue serait une mine, un drapeau sera posé.

Je pense que nous avons la base de la logique de notre IA. Nous nous baserons sur ce résonnement pour coder notre programme. Encore une foi, cette stratégie pourrait évoluer et changer selon l'avancement du projet.

Plusieurs fonctions seront nécessaires pour notre intelligence artificielle, commençons par celle qui analysera les voisines des voisines de nos cellules inconnues :

Fonction -> AnalyseVoisinesDesVoisines
In -> CoordXVoisine, CoordYVoisine
Out -> nbrCelluleVoisine, nbrCelluleVoisineDecouverte, nbrCelluleVoisineDrapeau

La fonction retournera le nombre des cellules voisine de la voisine passer en paramètre, le nombre de cellule découverte et le nombre de cellule avec un drapeau sur celle-ci. Elle prendra donc, en paramètre les coordonnées de la cellule voisine à laquelle nous analyseront ses voisines.

Une fonction pour activer et désactiver l'IA pendant la partie :

Fonction -> ActivationIA
In -> EstActiver
Out -> Null

La fonction prend en paramètre si l'IA est activée ou non, si elle est activée, la fonction la désactivera, dans le cas contraire elle activera l'IA qui se lancera automatiquement. Elle ne retourne aucune valeur.

Une fonction pour changer la vitesse d'action de l'intelligence artificielle (le même système que pour la difficulté du démineur), nous pourrions avoir 5 vitesses (très lent, lent, normal, rapide, très rapide et instantané). A savoir que la vitesse instantanée pourra entrainer de la latence plus ou moins grande selon la taille du plateau. En effet l'IA devra jouer en un minimum de temps, n'utilisant ni threads ni objets l'IA sera dite « **bloquante** » au niveau de l'application, il faudra attendre que l'IA termine avant de pouvoir interagir avec l'application.

Fonction -> ChengementVitesseIA
In -> VitesseActuelle
Out -> NouvelleVitesse

La fonction prend en paramètre la vitesse actuelle, que la fonction s'occupera de changer en fonction de cette vitesse actuelle. Elle retourne donc une nouvelle vitesse.

Enfin, une fonction pour appliquer les règles du démineur et concentrer toutes la logique, une fonction qui devra s'appeler elle-même toute les n seconde (déterminer par le temps entre chaque action de l'IA) jusqu'à se que la partie se termine ou que l'IA est désactivée.

Fonction -> ActionDeLIA
In -> Null
Out -> Null

La fonction ne prendra aucun paramètre ni ne retournera aucune valeur.