Synthèse du Projet d’ISN

Baccalauréat série S

2013 - 2014

Lycée George Dumézil, Vernon

Carpentier Valentin

***Synthèse du Projet d’ISN***

*Présentation du Projet :*

Dans le cadre du projet d’ISN pour la Baccalauréat, Mickaël Parlange et moi-même avons eu l’idée d’élaborer un jeu de bataille navale où une personne pourrait jouer contre une Intelligence Artificielle (IA). Pour cela nous avons dû penser à tout ce qui était nécessaire pour sa réalisation, que ce soit sur le plan graphique et pratique que sur le plan programmation. Ainsi, commençant à maitriser ce langage, nous avons décidé de faire notre jeu en Python et nous nous sommes aidés de la bibliothèque Pygame fournie par Python permettant de développer plus aisément des interfaces graphiques pour les jeux vidéo. Nous avons de plus analysé quatre principaux programmes qui devraient se trouver dans notre jeu pour qu’il puisse fonctionner : Un programme gérant le placement des navires du joueur et de l’IA, un programme gérant le tour de l’IA en deux parties. Une première où l’IA parcourrait le plateau à la recherche de bateaux et un second où l’IA tenterait de couler le navire qu’elle vient de trouver. Enfin, le dernier programme devait être celui gérant le tour du joueur, incluant le moment où ce dernier coule un navire pour l’avertir. Nous avons choisi de faire un plateau de vingt cases sur vingt pour y placer douze navires, trois navires de deux cases, trois navires de trois cases, trois navires de quatre cases, deux navire de cinq cases et un navire de six cases, pour créer un peu d’originalité par rapport à une strict bataille navale. Après avoir fait quelques tests avec Pygame pour comprendre son fonctionnement, nous nous sommes répartis le travail comme il suit : Je m’occupais du placement des navires du joueur et du tour de l’IA tandis que Mickaël s’occupait du placement des navires de l’IA et du tour du joueur.

Je ne détaillerais pas tout le programme et toutes les fonctions, celles-ci sont très nombreuses. Je commencerai par expliquer les bases du programme, les « CaseListes », puis j’expliquerai le fonctionnement du Main Programme avant de terminer sur la fonction de coulage des navires du joueur par l’IA

*I - Les Bases du Programmes de Bataille Navale : les « CaseListes » :*

Rapidement, nous avons constaté que l’affichage graphique, courant pour nous, serait incompréhensible pour l’ordinateur qui n’y voyait qu’une succession de pixel et non un véritable plateau de jeu comme nous. Il a fallut réfléchir à une alternative et nous avons eu l’idée d’affecter une variable booléenne à chaque case indiquant si un navire s’y trouvait (1 s’il y avait un bateau, 0 s’il n’y en avait pas). De plus, pour le programme de jeu de l’IA, une variable indiquant si une case a déjà été touchée nous semblait également indispensable.

Ainsi j’eu l’idée de mettre cette affectation sous forme d’une liste, une « CaseListe ». Le principe est simple, cette liste contient un certain nombre de termes tous lié à la même case. Le premier, sous forme d’un Tuple de deux termes, donne les coordonnées de la case. En effet, le programme ne voyant que des pixels dans la fenêtre, nous avons délimité des cases à vingt pixels chacune et leur position (marge par rapport au bord et écart pour le placement des navires et images type croix ou rond incluses) sert de lien entre notre perception et l’ordinateur. Ainsi, sur la grille du joueur, situé à 100 pixels de marge la marge gauche et haute, la case A1 est renseignée par le Tuple (105, 105) l’écart supplémentaire de 5 pixels étant pour permettre de placer un bateau correctement. De même, sur la deuxième grille, celle de l’IA, situé à 600 pixels d’écart à gauche et toujours 100 de haut, la case A1 est codé par son Tuple (605, 105). Il n’y a qu’une case qui a ses coordonnées et ce terme est ainsi la signature de la « CaseListe », la différenciant des autres. Le second terme, la variable « bateau/pas bateau », renseignant sur la présence d’un navire sur cette case. La troisième variable « touché/pas touché », renseigne l’ordinateur pour savoir si cette case a déjà été ciblé par un des joueurs au cours de la partie et permet d’avertir le joueur humain (sans lui permettre de recommencer cependant) ou d’éviter à l’IA de toucher deux fois la même case. La dernière variable, « vérifié/pas vérifié », permet lors du placement des navires, de savoir si la case a déjà été vérifié par le programme, ou, dans le cas de l’IA, de lui éviter de placer un nouveau navire à certains endroits.

En effet, pour s’assurer du bon fonctionnement de la partie, le programme de positionnement effectue un dernier test en fin de placement pour savoir si deux navires se touchent. Il utilise la variable « vérifié/pas vérifié » pour ne pas vérifié la position du même navire deux fois de suite et dans le cas de l’IA, bloque les cases du bateau nouvellement placé ainsi que toutes celles adjacentes pour que l’IA n’y place pas un navire et évite ainsi que deux navires se touchent.

Une « CaseListe » ressemble donc à ceci : [(105,105), 0, 0, 0]. Les zéro, présent par défaut, indiquant que la case ne possède pas de bateau, n’a pas été touché et n’a pas été vérifié ni affecter lors du placement des navires du joueur (car ici, l’abscisse 105 montre qu’il s’agit de la première grille, celle du joueur, situé à 100 pixels de marge à gauche).

Toutes les « CaseListes » sont répertoriés dans deux listes générales, la première, Grille1, rassemblant les 400 cases de la grille du joueur et la seconde, Grille2, rassemblant celle de la grille de l’IA. Ces grilles sont générées par une fonction *(cf. annexe 1)* en début de programme et constitues la base du programme sur lequel tout le jeu repose, que ce soit le placement des navires, le tour de l’IA ou du joueur, tous les programmes font appel à ses deux listes de 400 « CaseListes ».

*II - Présentation du fonctionnement du Main (cf. Annexe 3) :*

Gérant des différentes actions du jeu, le *Main\_Programme* a assez peu d’impact en lui-même, il sert surtout à gérer le graphisme avec Pygame et fait tourner indéfiniment une boucle de jeu. Il se décompose en deux parties, la partie initialisation, où les différents programmes sont importés, les listes Grille1 et Grille2 générées et la fenêtre de jeu initialisé (différents plateaux et textes de consigne).

La deuxième partie concerne le jeu en lui-même, une boucle tourne indéfiniment et fait le tour des différents évènements qui peuvent survenir (comme appuyez sur le clavier, utilisé la souris, vouloir fermer la fenêtre à l’aide de la croix rouge). Il y a cinq principales actions : quitter la partie en appuyant sur la croix, dans ce cas le programme demande confirmation avant de procédé à la sortie de la boucle si confirmation il y a eu.

Lorsque l’on se trouve dans la phase de placement des navires (renseigné par la variable booléenne « Initialisation ») et que l’on appuie sur la touche « espace », le programme procède au placement d’un nouveau navire par appel du programme *Fonction\_Ship* où toutes les fonctions relatives au placement des navires sont répertoriées. Il vérifie au préalable s’il y a déjà dix navires de placé et si c’est le cas, vérifie le positionnement des navires du joueur deux à deux (pour vérifier qu’aucun ne se touche) et lance le placement des navires de l’IA. S’il y a une erreur de positionnement, le programme stoppe tout et fait recommencer les saisies. Une fois le placement terminé, la variable « Initialisation » et mise à « False » et la variable « TourJoueur » (signalant que c’est au joueur de commencer) est mise à « True ».

Lorsque cette variable est à « True » et que le joueur appuie sur la flèche gauche, le tour du joueur est déclenché. Une fois celui-ci terminé, la variable « TourIA » est mise à « True » et lorsque le joueur appuie sur la flèche droite, l’IA joue son tour.

La dernière action est celle de la réinitialisation (pour recommencer une partie par exemple) lorsque l’on appuie sur la flèche du haut et déclenche (après confirmation) une fonction réinitialisant les deux grilles de jeu, les principales variables et le plateau de jeu pour redémarrer une partie. Une condition vérifie également l’état des variables de scores pour déterminé, en cas de nécessité, le vainqueur de la partie et stopper toutes les variables (empêchant le déclenchement des tours du joueur ou de l’IA), sans empêcher le joueur de réinitialiser la partie.

*III - Détaille d’un programme : le coulage d’un navire par l’IA :*

Enfin, je vais détailler le fonctionnement du programme qui permet à l’Intelligence Artificielle (IA) de couler les navires qu’elle vient de repérer. Cela ce fait en quatre temps :

Le premier est celui de repérer le navire, il utilise un programme (que je ne détaillerai pas ici) qui choisi une case du jeu, la touche et regarde si elle contient un navire. La plupart du temps il n’y a pas de bateau mais quand il y en a un, l’IA en prend compte dans une variable booléenne (InAttaque) qui lui servira lors de son prochain tour pour savoir qu’il lui faut couler le navire.

Le deuxième temps est celui de la détermination du sens du navire (cf. Annexe 4), un case est en effet insuffisante pour que l’IA se lance directement dans le coulage du navire sans un temps de recherche supplémentaire. Pour ce faire, l’IA utilise la même technique qu’un joueur lambda : le hasard. Ce choix permet de rendre l’IA moins difficile à battre car elle peut alors se tromper de sens et perdre un tour, comme le joueur (elle se trompe d’ailleurs souvent de sens, la chance est rarement avec elle). Un joueur le simule par son instinct et l’IA le simule par un compteur aléatoire entre 1 et 4 (le nombre de possibilité du sens du navire) et choisi sa case en conséquence. Les fonctionnements sont similaires d’un choix à l’autre et je ne prendrais que l’exemple du choix 1, hypothèse que le bateau continue sur la droite. Lorsque l’IA fait le choix numéro 1, il commence par vérifier l’existence de la case qu’il compte toucher, il peut effectivement avoir touché la dernière case de la ligne et cette vérification évite de nombreux bugs. Il vérifie ensuite que la case n’est pas déjà été touchée pour éviter de la retoucher. Si la case n’a pas été déjà touchée, l’IA touche la case et indique s’il a tiré dans l’eau ou sur le navire. S’il a touché le navire, l’IA considère que le navire continue sur la droite et retouchera à droite au prochain tour, sinon le navire ne continue plus sur la droite et l’IA se dirigera à gauche au prochain tour. L’intérêt des deux premiers tests, celui de l’existence de la case et du fait qu’elle soit déjà touchée, n’est pas que pour éviter les bugs ou les erreurs de l’IA mais permet aussi d’avoir des informations sur le navire touché. En effet, si la case à droite n’existe pas, le navire ne peut que continuer sur la gauche, c’est pour cela que l’IA le prend en compte et changera de direction d’attaque. De la même manière, si la case à droite à déjà été touché, ce ne pas être la case du navire, car si c’était une des cases du navire, l’IA aurait commencé à couler le navire par cette case, hors, il a commencé par une autre case donc cette case touchée est forcément dans l’eau. L’IA en déduit donc que le bateau se trouve maintenant à gauche et change immédiatement de direction.

Le troisième temps est celle du coulage proprement dit (cf. Annexe 5), une fois le sens du navire trouvé, l’IA reprend le principe de la phase deux mais en excluant la recherche de sens, car ce dernier est déterminé. Cependant, si la case se retrouve dans l’eau, a déjà été touché ou n’existe pas, l’IA change de direction mais conserve le sens du navire. Par exemple, si le choix 1 (navire à droite) s’est avéré juste et que, dans la phase trois, l’IA ne touche plus le navire en continuant à droite, il repartira à gauche (et non en haut ou en bas) car son sens a été déterminé comme étant horizontale quand une case à droite a été trouvé. Pour réussir le changement de sens et éviter les bugs, l’IA repart sur la première case touchée en cas de changement de sens, ainsi, les cases suivantes ne seront pas touchées. C’est en cela que l’IA est très efficace car elle anticipe de nombreuses actions, qui peuvent paraître instinctive pour un joueur, mais qui ne le sont pas du tout pour une machine (sachant qu’il faut prendre en compte que celle-ci n’a pas conscience qu’elle joue à un jeu de Bataille Navale). Cette méthode permet de trouver toutes les cases d’un navire très simplement, du moment que les bateaux sont bien positionnés, qu’aucun ne se touche. Car si deux bateaux se touchent, l’IA peut confondre les deux navires, il sera en effet impossible de déterminé quelle case appartient à quelle navire, rendant le jeu ingérable pour l’IA.

Mais cette méthode comporte un problème, l’IA cherche des cases d’un navire et les touche une par une, changeant de sens dès qu’elle touche dans l’eau, se retrouve face à une case déjà touchée ou une case inexistante, mais jamais elle ne connait le nombre de case du navire. La première solution aurait été de lui faire une tour de plus, ou l’IA touche une dernière fois dans l’eau, et se rend compte que les deux extrémités étant atteinte, le navire est coulé, mais cela lui aurait fait perdre un tour. En effet, dans une partie entre deux joueurs, ce n’est pas à l’attaquant de deviner quand le navire est coulé mais au défenseur de le signaler une fois la dernière case touchée. L’IA le fait quand le joueur lui coule un bateau mais le joueur ne donne aucune information de ce type à l’IA (c’est le programme qui gère les informations de cases, le joueur ne dit rien), il faut donc une variable qui donne le nombre de case du navire en train d’être coulé. C’est le rôle de la fonction *CountNbCoulIANavire* (*cf. Annexe 4*), s’inspirant de la fonction *Coulage* de Mickaël, elle met dans une liste toute les cases comportant un bateau adjacente les unes aux autres, en partant de la case touché, et en déduit le nombre de case du navire (le nombre de case trouvé ainsi plus celle déjà touché donne le nombre de case du navire). En comparant ce nombre au nombre de coup porté au bateau (incrémenté à chaque touché de l’IA), le programme détermine si un navire est coulé. Le score est incrémenté et le programme passe dans une dernière phase.

Le quatrième temps est celui du Checkage des cases autour du navire. Deux navires ne pouvant se toucher, l’IA va marquer les cases autour du navire qui vient d’être touché pour éviter de les toucher plus tard, étant donné qu’aucun navire ne s’y trouvera (cela renforce l’efficacité du joueur virtuel). Pour ce faire, elle va partir de la case la plus à gauche du navire (ou la plus en haut en fonction du sens), Le programme commence par vérifier l’existence des cases adjacentes et les inscrits dans des variables booléennes tel que « case\_droite » (True veut dire qu’il y a une case à droite et False veut dire qu’il n’y en a pas) et met à 1 la variable « touché/pas touché » de toutes les cases autour de cette dernière, si elles existent (*cf. Annexe 5 et 7*). Une fois ceci fait, le programme enchaîne sur une autre fonction qui va mettre à 1 la variable « touché/pas touché » de toutes les autres cases adjacentes au navire en fonction de son sens en fonctionnant ainsi : On prend la case suivante, on « touche » les deux cases en dessus et en dessous (ou à droite et à gauche si le navire est vertical) et on fait ceci jusqu’à arriver à la fin du navire. Dans ce cas, on met à 1 la case suivante ainsi que celle au dessus et en dessous (ou à droite et à gauche) (*cf. Annexe 6 et 7*). Le navire est maintenant coulé et l’IA ne touchera pas les cases adjacentes, elle peut reprendre sa recherche pour trouver un nouveau navire (ou annoncer sa victoire si c’était le dernier navire du joueur).

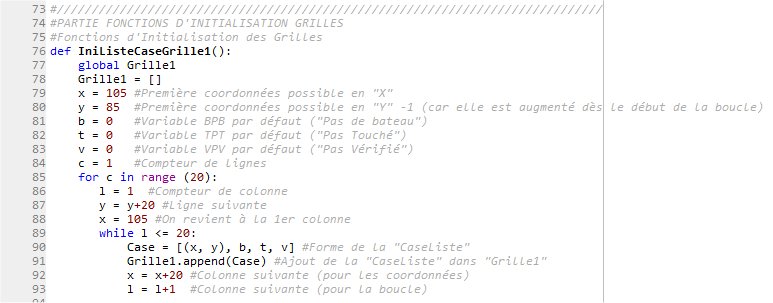
*Conclusion du Projet :*

En conclusion, ce projet de bataille navale, dont le principe peu paraître assez banal, nous a montré les différentes difficultés de l’informatique. Mais nous les avons surmontés et avons terminé notre programme dans les temps et en en excluant tous bugs. La difficulté majeure fut d’accorder la vision du jeu par le joueur humain et la vision par l’ordinateur qui sont radicalement différents. Le joueur voyant des grilles codés de A à T et de 1 à 20 où se trouvent des bateaux qu’il tente de couler alors que l’ordinateur voit une liste de listes à variables dans laquelle il fait des tests et dont les actions dépendent de la valeur de ses variables, le résultat donnant une impression de jeu.

Cependant, notre travail en équipe autonome et efficace nous a permis de résoudre rapidement ses problèmes pour finir le projet dans les temps. Le plus gros du travail demeure le placement des navires, étape crucial pour le bon déroulement du jeu. La suite fut plus rapide pour deux raisons, elle demandait moins de travail et nous étions plus expérimentés dans la programmation en python.

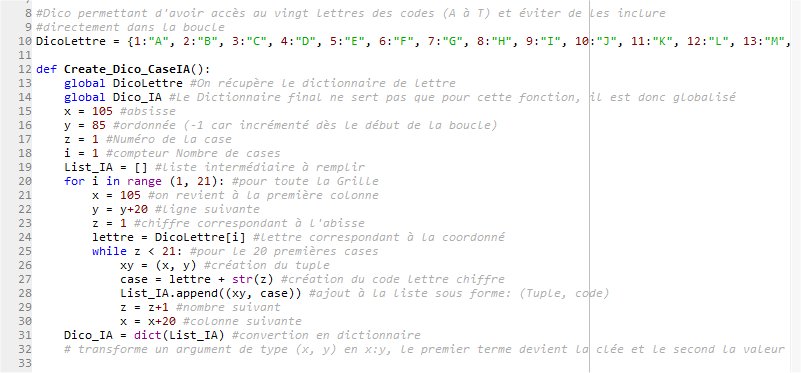
Enfin, étant en avance sur la date limite du projet, nous avons ajouté une fonction non prévu à la base qui permet à l’Intelligence Artificielle (IA) de « dialoguer » avec le joueur en cours de partie. En effet, lorsque l’IA ou le joueur termine leur tour, l’IA peut écrire, dans une chance sur trois en moyenne grâce à une simulation de probabilité, une phrase amusante en haut du plateau pour se mettre en valeur et tenter de dévaloriser le joueur. Cette fonction a plus pour but de rendre une partie amusante que de rendre le jeu jouable. Développé par Mickaël Parlange, cette fonction fonctionne sans problème, son seul défaut serait le manque de réplique de l’ordinateur créant de nombreuses répétitions de l’IA au cours de la partie.

*Annexe N°01 : Fonction de Génération de la Liste Grille1*



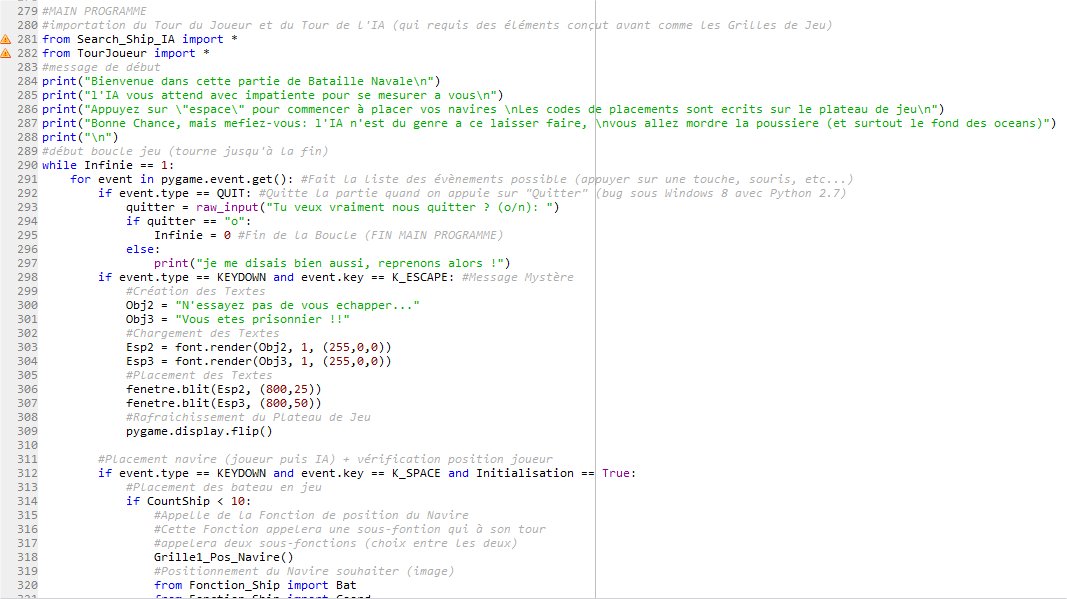
*Annexe N°02 : Fonction de Génération du Dictionnaire Dico\_IA*

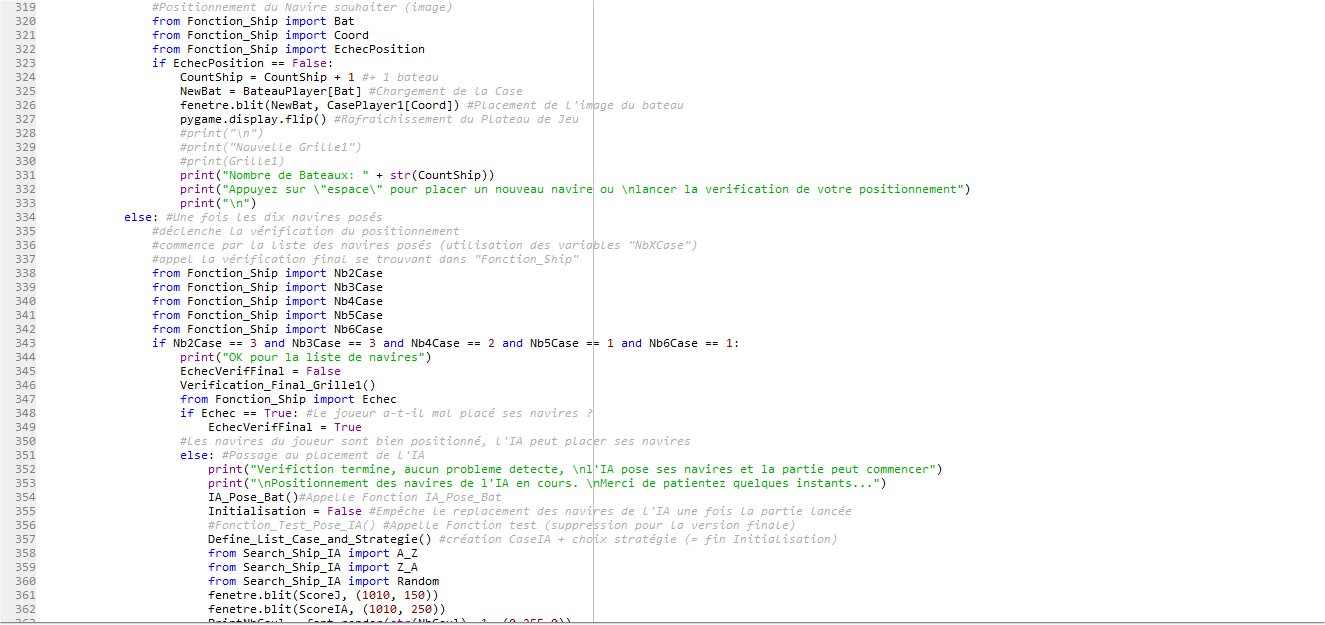
Dico\_IA est le dictionnaire qui permet, pour un Tuple de coordonnée donné, de ressortir la case correspondant (par exemple « A1 »), il sert pour l’affichage des cases touchées par l’IA :

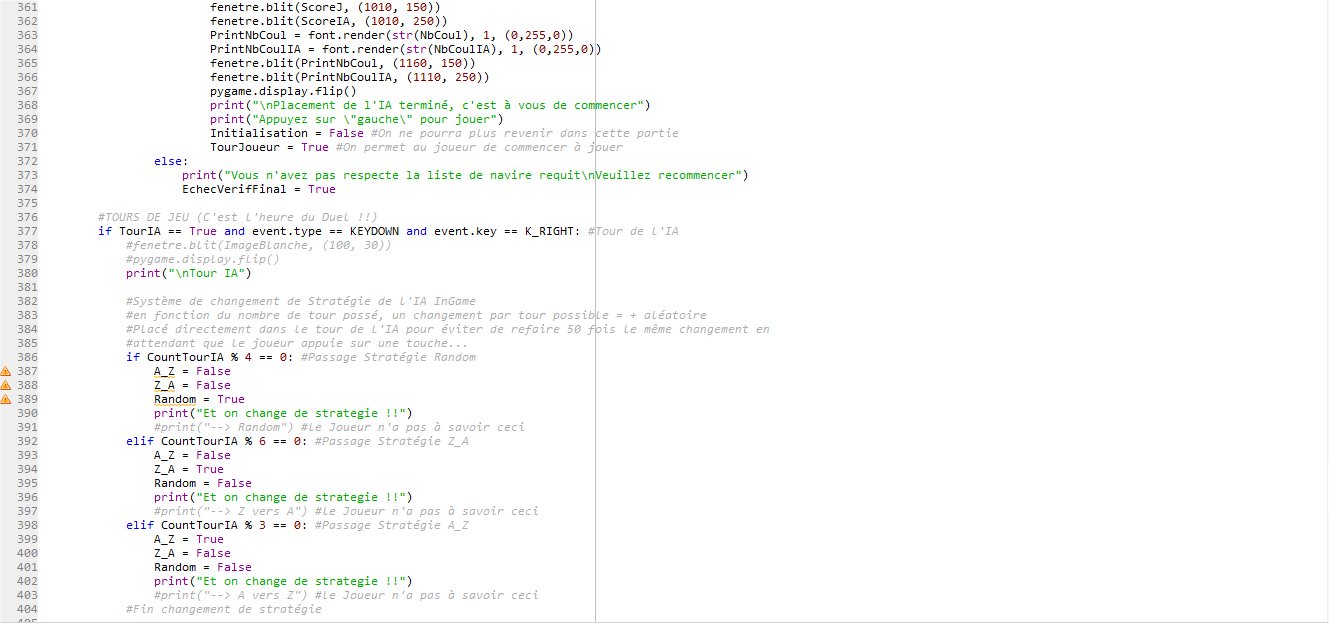


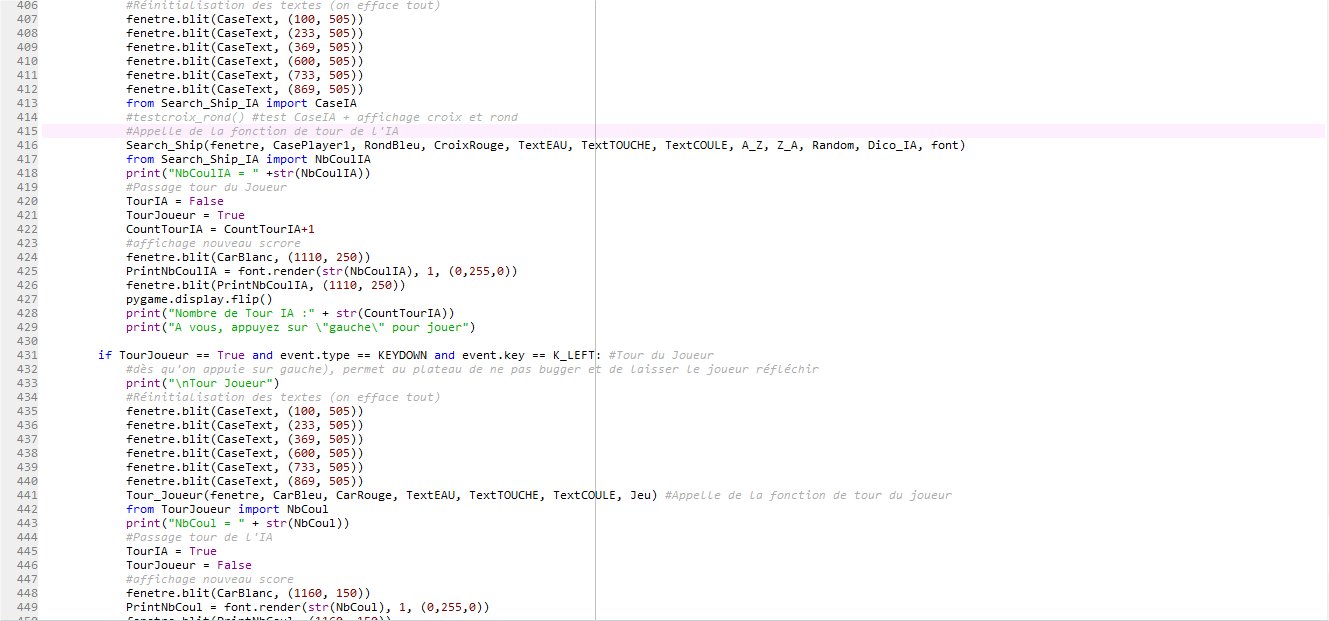
*Annexe N°03 : Main\_Programme, partie gestion du jeu*

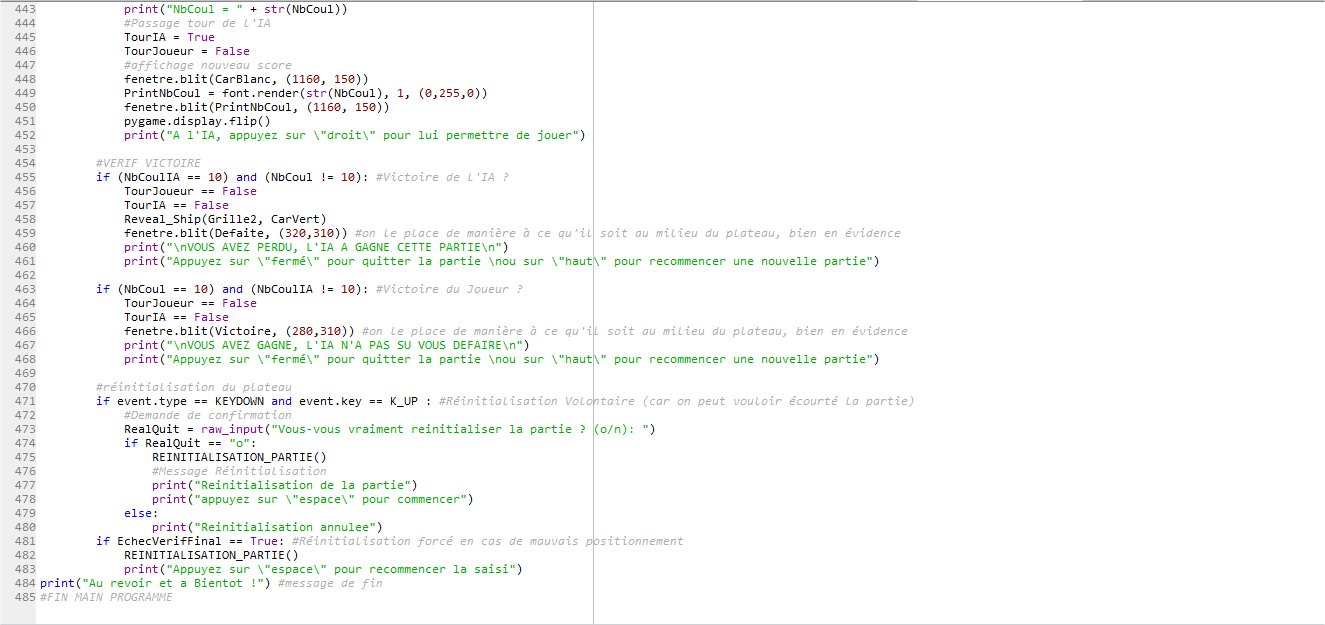
L’Initialisation est absente :



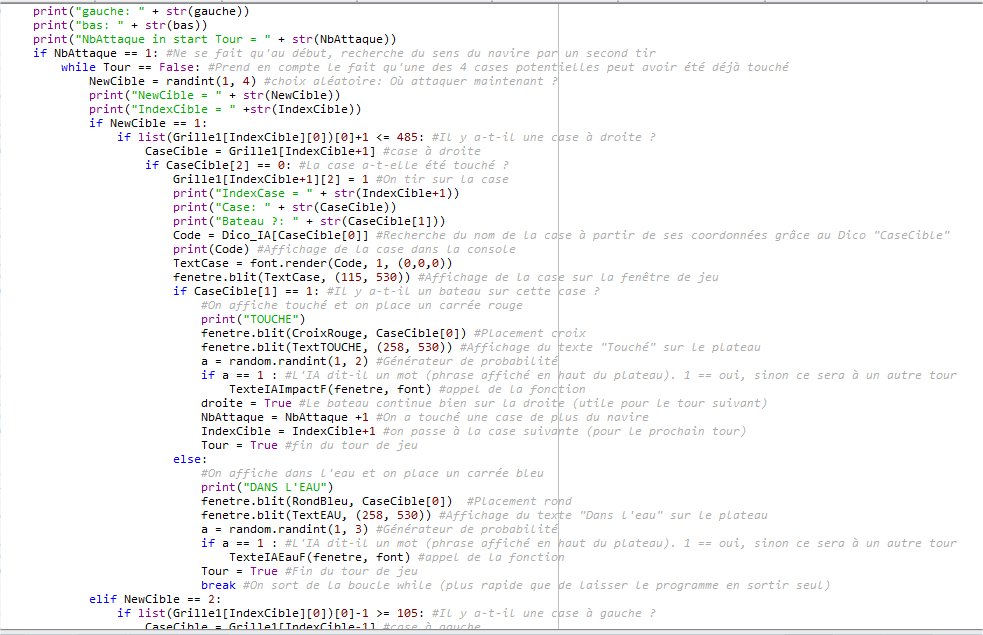


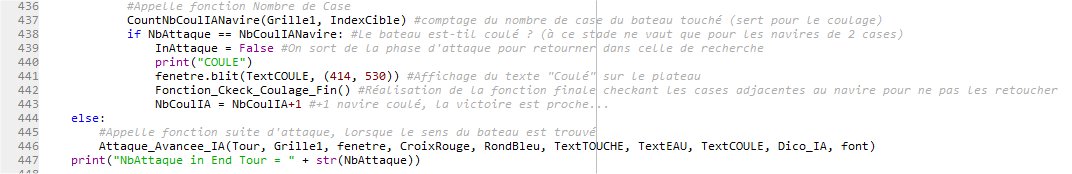




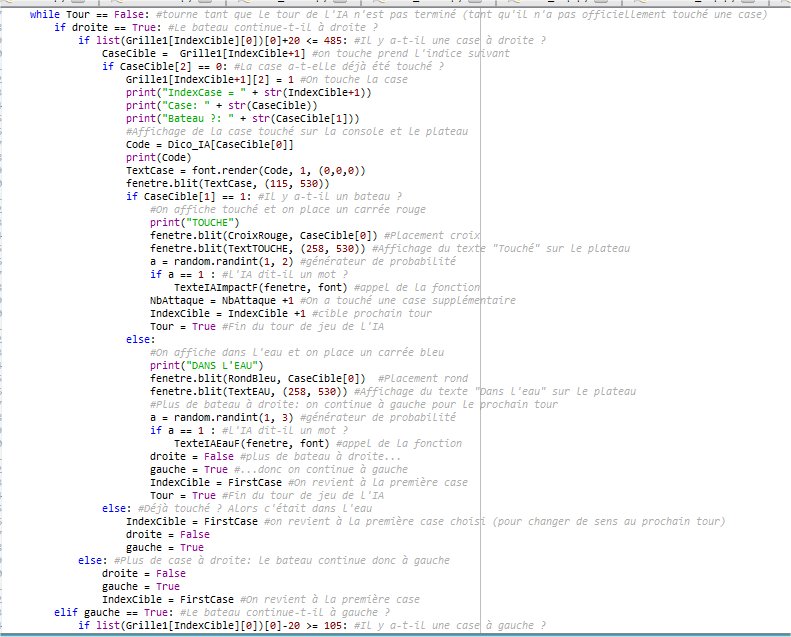


*Annexe N°04 : Exemple d’Attaque programme de coulage Phase I + Vérification du coulage du navire*

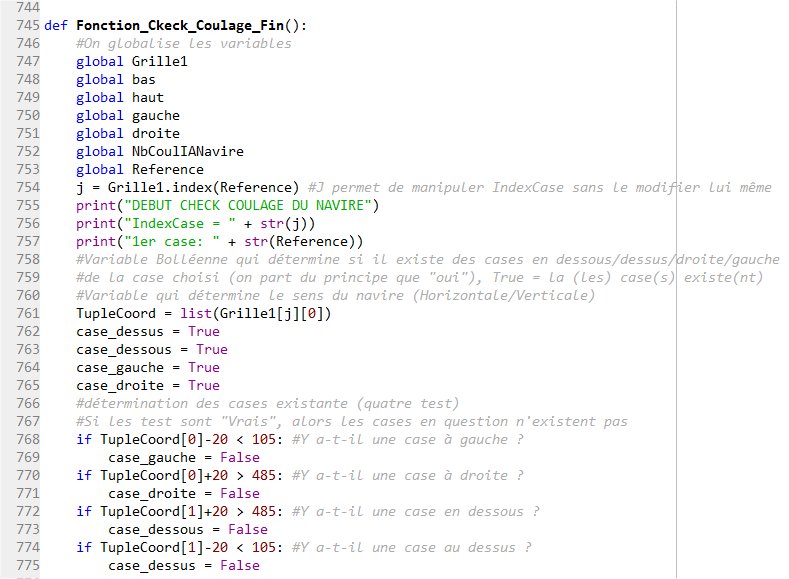


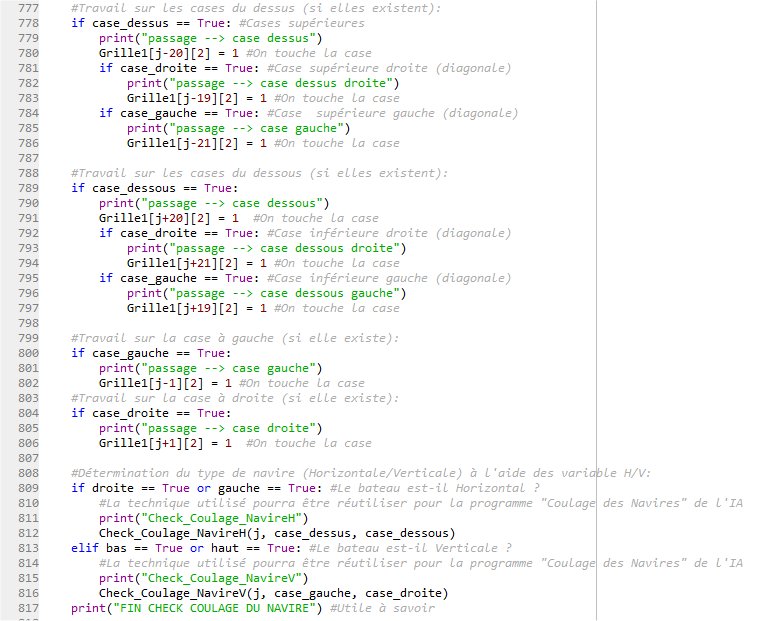


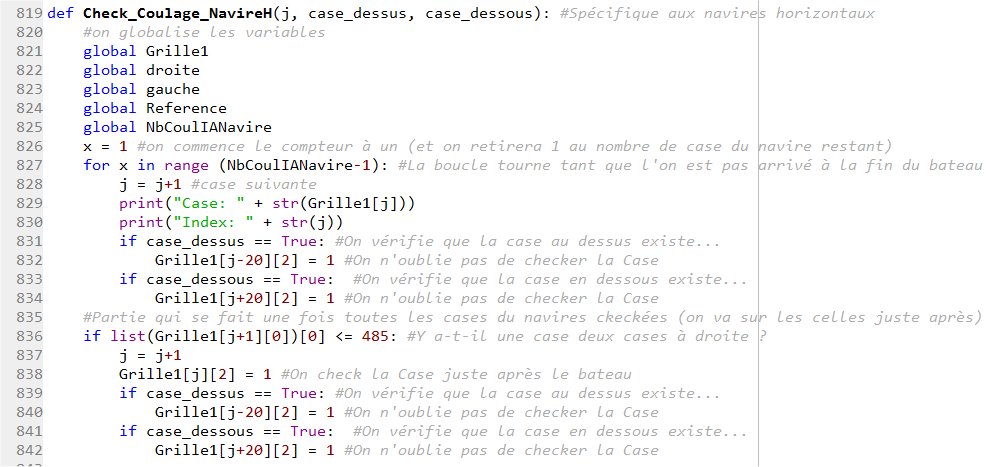
*Annexe N°05 : Exemple d’Attaque programme de coulage Phase II*



*Annexe N°06 : Programme de Coulage Phase IV (exemple pour un navire Horizontal)*





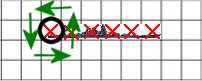


*Annexe N°07 : Coulage Phase IV, images explicatives*

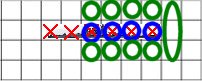
Le programme de checkage des cases fonctionnent de la sorte. Tout d’abord, le navire est coulé :



La première fonction (*Fonction\_Check\_Coulage\_Fin*) touche les cases autour de la plus à gauche (en noir) comme le montre les flèches vertes



Enfin, la deuxième fonction (*Check\_Coulage\_NavireH*) se positionne sur les cases suivantes (en bleu) et touche celle au dessus et en dessous (en vert) avant de terminé par les trois après le bateau (ellipse verte)



Le principe est le même si le navire est vertical, sauf que les cases touchées seront à droite et à gauche et que la case suivante sera en dessous et non à droite.