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 le 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,
- deux programmes gérant le tour de l'IA : un premier où l'IA parcourt le plateau à la recherche de bateaux et un second où l'IA tente de couler le navire qu'elle vient de trouver,
- enfin, le dernier programme doit 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 avec deux grilles de jeu de 20 cases sur 20 pour y placer 12 navires. Sur l'écran, la grille de gauche est celle du joueur, celle de droite, la grille de l'IA (*cf. Annexe 1*).

Navires	3	3	3	2	1
Cases	2	3	4	5	6

Cette disposition crée un peu d'originalité par rapport à une stricte bataille navale. Après avoir fait quelques tests avec Pygame pour comprendre son fonctionnement, nous nous sommes répartis le travail comme 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 détaillerai 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 Programme 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 pixels et non un véritable plateau de jeu comme nous le voyons. 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'eus 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 liés à la même case. Dans la version actuelle, il y a 4 termes : 1 « tuple » et 3 variables booléennes. Le 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 de vingt pixels chacune et leur position sert de lien entre le joueur (ce qu'il perçoit) et l'ordinateur ; position = marge par rapport au bord du plateau et écart pour le placement des navires dans la grille (images type croix ou rond incluses) (*cf. Annexe 1*). Ainsi, sur la grille du joueur (à gauche), située à 100 pixels de marge à gauche et en haut du plateau, la case A1 est renseignée par le tuple (105, 105), l'écart supplémentaire de 5 pixels permettant de placer un bateau sans masquer les contours de la case. De même, sur la deuxième grille, celle de l'IA (à droite), située à 600 pixels de marge à gauche et toujours 100 pixels de marge en haut, la case A1 est codée par son Tuple (605, 105). Il n'y a qu'une case qui a ces coordonnées donc ce tuple est la signature de la « CaseListe », la différenciant des autres. Le second terme est la variable « bateau/pas bateau » renseignant sur la présence d'un navire sur cette case. Le troisième terme est la variable « touché/pas touché », qui renseigne l'ordinateur pour savoir si cette case a déjà été ciblée par un des joueurs au cours de la partie : avertit le joueur humain qu'il a déjà touché cette case (sans lui permettre de recommencer cependant) et évite à l'IA de toucher deux fois la même case. Le dernier terme, la dernière variable, « vérifié/pas vérifié », permet lors du placement des navires, de savoir si la case a déjà été vérifiée 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 ou non. Il utilise la variable « vérifié/pas vérifié » pour ne pas vérifier 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]. L'abscisse 105 montre qu'il s'agit de la première grille, celle du joueur, situé à 100 pixels de marge à gauche du plateau. Les zéros, présents par défaut, indiquent que la case ne possède pas de bateau, n'a pas été touché et n'a pas été vérifiée ni affectée lors du placement des navires du joueur. Toutes les « CaseListes » sont répertoriées dans deux listes générales, la première, « Grille1 », rassemblant les 400 cases de la grille du joueur et la seconde, « Grille2 », rassemblant celles de la grille de l'IA. Ces grilles sont générées par une fonction *(cf. annexe 2)* en début de programme et constituent 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 les 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 première est la partie initialisation, où les différents programmes sont importés, les listes Grille1 et Grille2 générées et la fenêtre de jeu initialisée (différentes grilles 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 « appuyer sur le clavier », « utiliser la souris », « vouloir fermer la fenêtre à l'aide de la croix rouge »). Il y a cinq principales actions : quitter la partie, placer les navires, permettre le tour du joueur, permettre le tour de l'IA et réinitialiser la partie.

En appuyant sur la croix pour quitter la partie, le programme demande confirmation avant de procéder à la sortie de la boucle si le joueur a confirmé.

Lorsqu'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 douze navires de placés, si c'est le cas, il 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 » est 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. Il 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 déterminer, à la fin du jeu, 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 - Détail d'un programme : le coulage d'un navire par l'IA :

Dans ce paragraphe, 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 temps est celui du repérage du navire. Il utilise un programme (que je ne détaillerai pas ici) qui choisit 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*). Une 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 possibilités 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'a 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 touchée, 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 touchée 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 horizontal quand une case à droite a été trouvée. 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 instinctives 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 et qu'aucun ne se touche. Car si deux bateaux se touchent, l'IA peut confondre les deux navires, il sera en effet impossible de déterminer quelle case appartient à quel 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 cases du navire. La première solution aurait été de lui faire un tour de plus, où l'IA touche une dernière fois dans l'eau, et se rend compte que les deux extrémités étant atteintes, 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 cases 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 adjacentes les unes aux autres, en partant de la case touchée, et en déduit le nombre de cases du navire (le nombre de cases trouvé ainsi plus celle déjà touchée donne le nombre de cases du navire). En

comparant ce nombre au nombre de coups portés 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 trouve (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 inscrit 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 6 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).

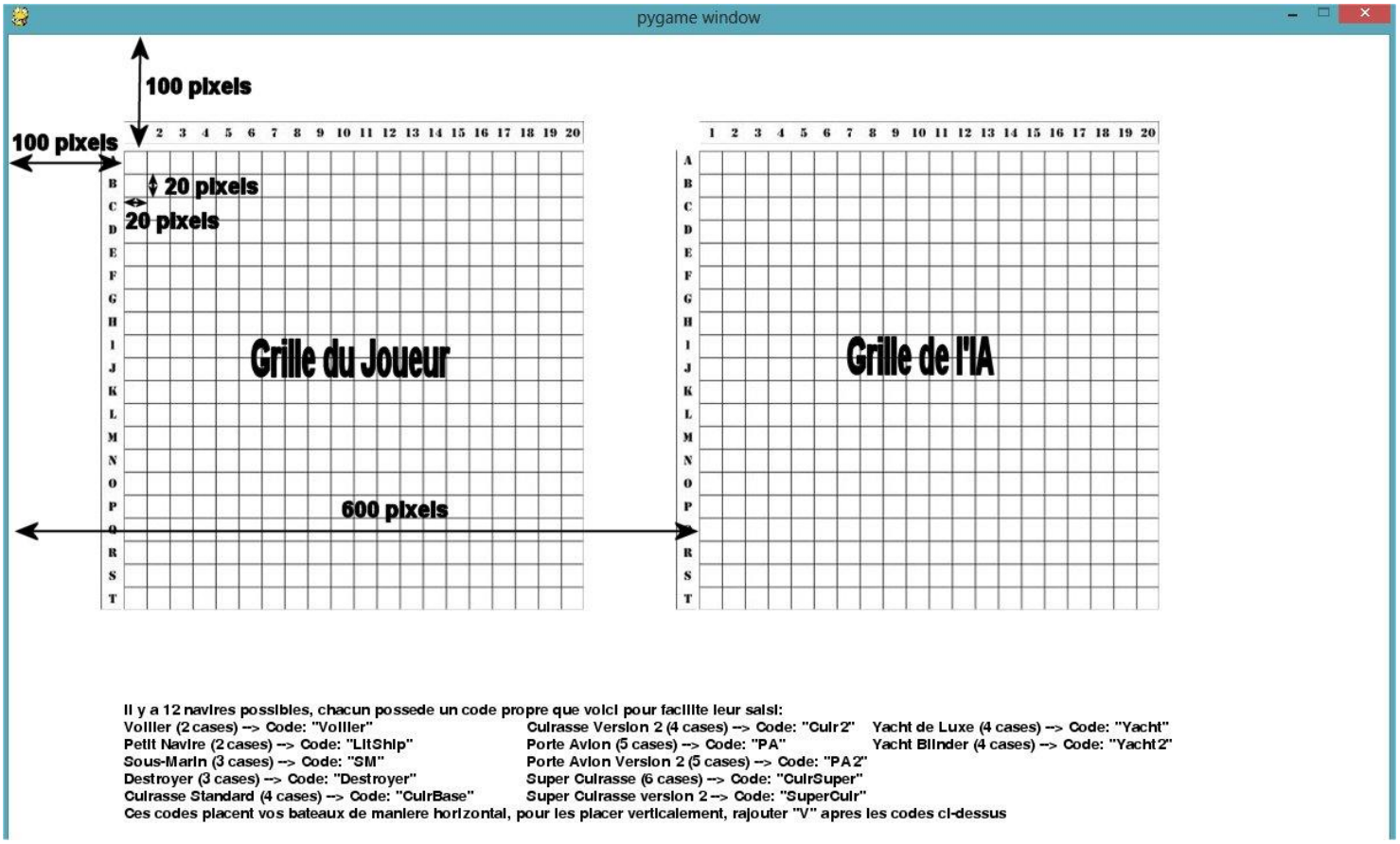
Conclusion du Projet :

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 tout bug. La difficulté majeure fut d'accorder la vision du jeu par le joueur humain et la vision par l'ordinateur qui sont radicalement différentes. Le joueur voyant des grilles codées 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 série de listes de variables sur lesquelles il fait des tests et dont les actions dépendent de la valeur de ces variables, le résultat donnant une impression de jeu.

Cependant, notre travail en équipe autonome et efficace nous a permis de résoudre rapidement ces problèmes pour finir le projet dans les temps. Le plus gros du travail demeure le placement des navires, étape cruciale 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 prévue 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 son 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 donner une touche amusante au jeu que de participer à le rendre jouable. Développé par Mickaël Parlange, cette fonction fonctionne sans problème, sa limite serait le manque de réplique de l'ordinateur créant de nombreuses répétitions de l'IA au cours de la partie, limite qui peut être aisément corrigée.

Annexe N°1 : Capture d'écran de l'écran de jeu



Annexe N°2 : Fonction de Génération de la Liste Grille1

```
73 #////////////////////////////////////////////////////////////////////////////////
74 #PARTIE FONCTIONS D'INITIALISATION GRILLES
75 #Fonctions d'Initialisation des Grilles
76 def IniListeCaseGrille1():
77     global Grille1
78     Grille1 = []
79     x = 105 #Première coordonnées possible en "X"
80     y = 85  #Première coordonnées possible en "Y" -1 (car elle est augmenté dès le début de la boucle)
81     b = 0   #Variable BPB par défaut ("Pas de bateau")
82     t = 0   #Variable TPT par défaut ("Pas Touché")
83     v = 0   #Variable VPV par défaut ("Pas Vérifié")
84     c = 1   #Compteur de lignes
85     for c in range (20):
86         l = 1  #Compteur de colonne
87         y = y+20 #Ligne suivante
88         x = 105 #On revient à la 1er colonne
89         while l <= 20:
90             Case = [(x, y), b, t, v] #Forme de la "CaseListe"
91             Grille1.append(Case) #Ajout de la "CaseListe" dans "Grille1"
92             x = x+20 #Colonne suivante (pour les coordonnées)
93             l = l+1  #Colonne suivante (pour la boucle)
```

Annexe N°3 : Main Programme, partie gestion du jeu

L'Initialisation est absente :

```
#MAIN PROGRAMME
#importation du Tour du Joueur et du Tour de l'IA (qui requis des éléments conçut avant comme les Grilles de Jeu)
from Search_Ship_IA import *
from TourJoueur import *
#message de début
print("Bienvenue dans cette partie de Bataille Navale\n")
print("l'IA vous attend avec impatiente pour se mesurer a vous\n")
print("Appuyez sur \"espace\" pour commencer à placer vos navires \nLes codes de placements sont ecrits sur le plateau de jeu\n")
print("Bonne Chance, mais mefiez-vous: l'IA n'est du genre a ce laisser faire, \nvous allez mordre la poussiere (et surtout le fond des oceans)")
print("\n")
#début boucle jeu (tourne jusqu'à la fin)
while Infinie == 1:
    for event in pygame.event.get(): #Fait la liste des évènements possible (appuyer sur une touche, souris, etc...)
        if event.type == QUIT: #Quitte la partie quand on appuie sur "Quitter" (bug sous Windows 8 avec Python 2.7)
            quitter = raw_input("Tu veux vraiment nous quitter ? (o/n): ")
            if quitter == "o":
                Infinie = 0 #Fin de la Boucle (FIN MAIN PROGRAMME)
            else:
                print("je me disais bien aussi, reprenons alors !")
        if event.type == KEYDOWN and event.key == K_ESCAPE: #Message Mystère
            #Création des Textes
            Obj2 = "N'essayez pas de vous echapper..."
            Obj3 = "Vous etes prisonnier !!"
            #Chargement des Textes
            Esp2 = font.render(Obj2, 1, (255,0,0))
            Esp3 = font.render(Obj3, 1, (255,0,0))
            #Placement des Textes
            fenetre.blit(Esp2, (800,25))
            fenetre.blit(Esp3, (800,50))
            #Rafraichissement du Plateau de Jeu
            pygame.display.flip()

        #Placement navire (joueur puis IA) + vérification position joueur
        if event.type == KEYDOWN and event.key == K_SPACE and Initialisation == True:
            #Placement des bateau en jeu
            if CountShip < 10:
                #Appelle de la Fonction de position du Navire
                #Cette Fonction appelera une sous-fontion qui à son tour
                #appelera deux sous-fonctions (choix entre les deux)
                Grille1_Pos_Navire()
                #Positionnement du Navire souhaiter (image)
                from Fonction_Ship import Bat
                from Fonction_Ship import Coord
                from Fonction_Ship import EchecPosition
                if EchecPosition == False:
                    CountShip = CountShip + 1 #+ 1 bateau
                    NewBat = BateauPlayer[Bat] #Chargement de la Case
                    fenetre.blit(NewBat, CasePlayer1[Coord]) #Placement de l'image du bateau
                    pygame.display.flip() #Rafraichissement du Plateau de Jeu
                    #print("\n")
                    #print("Nouvelle Grille1")
                    #print(Grille1)
                    print("Nombre de Bateaux: " + str(CountShip))
                    print("Appuyez sur \"espace\" pour placer un nouveau navire ou \nlancer la verification de votre positionnement")
                    print("\n")
            else: #Une fois les dix navires posés
                #déclenche la vérification du positionnement
                #commence par la liste des navires posés (utilisation des variables "NbXCase")
                #appel la vérification final se trouvant dans "Fonction_Ship"
                from Fonction_Ship import Nb2Case
                from Fonction_Ship import Nb3Case
                from Fonction_Ship import Nb4Case
                from Fonction_Ship import Nb5Case
                from Fonction_Ship import Nb6Case
                if Nb2Case == 3 and Nb3Case == 3 and Nb4Case == 2 and Nb5Case == 1 and Nb6Case == 1:
                    print("OK pour la liste de navires")
                    EchecVerifFinal = False
                    Verification_Final_Grille1()
                    from Fonction_Ship import Echec
                    if Echec == True: #Le joueur a-t-il mal placé ses navires ?
                        EchecVerifFinal = True
                    #Les navires du joueur sont bien positionné, l'IA peut placer ses navires
                    else: #Passage au placement de l'IA
                        print("Verifiction termine, aucun probleme detecte, \nl'IA pose ses navires et la partie peut commencer")
                        print("\nPositionnement des navires de l'IA en cours. \nMerci de patientez quelques instants...")
                        IA_Pose_Bat()#Appelle Fonction IA_Pose_Bat
                        Initialisation = False #Empêche le replacement des navires de l'IA une fois la partie lancée
                        #Fonction_Test_Pose_IA() #Appelle Fonction test (suppression pour la version finale)
                        Define_List_Case_and_Strategie() #création CaseIA + choix stratégie (= fin Initialisation)
                        from Search_Ship_IA import A_Z
                        from Search_Ship_IA import Z_A
                        from Search_Ship_IA import Random
                        fenetre.blit(ScoreJ, (1010, 150))
                        fenetre.blit(ScoreIA, (1010, 250))
```

```
                    fenetre.blit(ScoreJ, (1010, 150))
                    fenetre.blit(ScoreIA, (1010, 250))
                    PrintNbCoul = font.render(str(NbCoul), 1, (0,255,0))
                    PrintNbCoulIA = font.render(str(NbCoulIA), 1, (0,255,0))
                    fenetre.blit(PrintNbCoul, (1160, 150))
                    fenetre.blit(PrintNbCoulIA, (1110, 250))
                    pygame.display.flip()
                    print("\nPlacement de l'IA terminé, c'est à vous de commencer")
                    print("Appuyez sur \"gauche\" pour jouer")
                    Initialisation = False #On ne pourra plus revenir dans cette partie
                    TourJoueur = True #On permet au joueur de commencer à jouer
            else:
                print("Vous n'avez pas respecte la liste de navire requit\nVeuillez recommencer")
                EchecVerifFinal = True

    #TOURS DE JEU (C'est l'heure du Duel !!)
    if TourIA == True and event.type == KEYDOWN and event.key == K_RIGHT: #Tour de l'IA
        #fenetre.blit(ImageBlanche, (100, 30))
        #pygame.display.flip()
        print("\nTour IA")

        #Système de changement de Stratégie de l'IA InGame
        #en fonction du nombre de tour passé, un changement par tour possible = + aléatoire
        #Placé directement dans le tour de l'IA pour éviter de refaire 50 fois le même changement en
        #attendant que le joueur appuie sur une touche...
        if CountTourIA % 4 == 0: #Passage Stratégie Random
            A_Z = False
            Z_A = False
            Random = True
            print("Et on change de strategie !!")
            #print("--> Random") #Le Joueur n'a pas à savoir ceci
        elif CountTourIA % 6 == 0: #Passage Stratégie Z_A
            A_Z = False
            Z_A = True
            Random = False
            print("Et on change de strategie !!")
            #print("--> Z vers A") #Le Joueur n'a pas à savoir ceci
        elif CountTourIA % 3 == 0: #Passage Stratégie A_Z
            A_Z = True
            Z_A = False
            Random = False
            print("Et on change de strategie !!")
            #print("--> A vers Z") #Le Joueur n'a pas à savoir ceci
        #Fin changement de stratégie
```

```
        #Réinitialisation des textes (on efface tout)
        fenetre.blit(CaseText, (100, 505))
        fenetre.blit(CaseText, (233, 505))
        fenetre.blit(CaseText, (369, 505))
        fenetre.blit(CaseText, (600, 505))
        fenetre.blit(CaseText, (733, 505))
        fenetre.blit(CaseText, (869, 505))
        from Search_Ship_IA import CaseIA
        #testcroix_rond() #test CaseIA + affichage croix et rond
        #Appelle de la fonction de tour de l'IA
        Search_Ship(fenetre, CasePlayer1, RondBleu, CroixRouge, TextEAU, TextTOUCHE, TextCOULE, A_Z, Z_A, Random, Dico_IA, font)
        from Search_Ship_IA import NbCoulIA
        print("NbCoulIA = " +str(NbCoulIA))
        #Passage tour du Joueur
        TourIA = False
        TourJoueur = True
        CountTourIA = CountTourIA+1
        #affichage nouveau scrore
        fenetre.blit(CarBlanc, (1110, 250))
        PrintNbCoulIA = font.render(str(NbCoulIA), 1, (0,255,0))
        fenetre.blit(PrintNbCoulIA, (1110, 250))
        pygame.display.flip()
        print("Nombre de Tour IA :" + str(CountTourIA))
        print("A vous, appuyez sur \"gauche\" pour jouer")

    if TourJoueur == True and event.type == KEYDOWN and event.key == K_LEFT: #Tour du Joueur
        #dès qu'on appuie sur gauche), permet au plateau de ne pas bugger et de laisser le joueur réfléchir
        print("\nTour Joueur")
        #Réinitialisation des textes (on efface tout)
        fenetre.blit(CaseText, (100, 505))
        fenetre.blit(CaseText, (233, 505))
        fenetre.blit(CaseText, (369, 505))
        fenetre.blit(CaseText, (600, 505))
        fenetre.blit(CaseText, (733, 505))
        fenetre.blit(CaseText, (869, 505))
        Tour_Joueur(fenetre, CarBleu, CarRouge, TextEAU, TextTOUCHE, TextCOULE, Jeu) #Appelle de la fonction de tour du joueur
        from TourJoueur import NbCoul
        print("NbCoul = " + str(NbCoul))
        #Passage tour de l'IA
        TourIA = True
        TourJoueur = False
        #affichage nouveau score
        fenetre.blit(CarBlanc, (1160, 150))
        PrintNbCoul = font.render(str(NbCoul), 1, (0,255,0))
```

```
443             print("NbCoul = " + str(NbCoul))
444             #Passage tour de l'IA
445             TourIA = True
446             TourJoueur = False
447             #affichage nouveau score
448             fenetre.blit(CarBlanc, (1160, 150))
449             PrintNbCoul = font.render(str(NbCoul), 1, (0,255,0))
450             fenetre.blit(PrintNbCoul, (1160, 150))
451             pygame.display.flip()
452             print("A l'IA, appuyez sur \"droit\" pour lui permettre de jouer")
453 
454         #VERIF VICTOIRE
455         if (NbCoulIA == 10) and (NbCoul != 10): #Victoire de l'IA ?
456             TourJoueur == False
457             TourIA == False
458             Reveal_Ship(Grille2, CarVert)
459             fenetre.blit(Defaite, (320,310)) #on le place de manière à ce qu'il soit au milieu du plateau, bien en évidence
460             print("\nVOUS AVEZ PERDU, L'IA A GAGNE CETTE PARTIE\n")
461             print("Appuyez sur \"fermé\" pour quitter la partie \nou sur \"haut\" pour recommencer une nouvelle partie")
462 
463         if (NbCoul == 10) and (NbCoulIA != 10): #Victoire du Joueur ?
464             TourJoueur == False
465             TourIA == False
466             fenetre.blit(Victoire, (280,310)) #on le place de manière à ce qu'il soit au milieu du plateau, bien en évidence
467             print("\nVOUS AVEZ GAGNE, L'IA N'A PAS SU VOUS DEFAIRE\n")
468             print("Appuyez sur \"fermé\" pour quitter la partie \nou sur \"haut\" pour recommencer une nouvelle partie")
469 
470         #réinitialisation du plateau
471         if event.type == KEYDOWN and event.key == K_UP : #Réinitialisation Volontaire (car on peut vouloir écourté la partie)
472             #Demande de confirmation
473             RealQuit = raw_input("Vous-vous vraiment reinitialiser la partie ? (o/n): ")
474             if RealQuit == "o":
475                 REINITIALISATION_PARTIE()
476                 #Message Réinitialisation
477                 print("Reinitialisation de la partie")
478                 print("appuyez sur \"espace\" pour commencer")
479             else:
480                 print("Reinitialisation annulee")
481         if EchecVerifFinal == True: #Réinitialisation forcé en cas de mauvais positionnement
482             REINITIALISATION_PARTIE()
483             print("Appuyez sur \"espace\" pour recommencer la saisi")
484 print("Au revoir et a Bientot !") #message de fin
485 #FIN MAIN PROGRAMME
```

Annexe N°4 : Exemple d'Attaque programme de coulage Phase I + Vérification du coulage du navire

```
print("gauche: " + str(gauche))
print("bas: " + str(bas))
print("NbAttaque in start Tour = " + str(NbAttaque))
if NbAttaque == 1: #Ne se fait qu'au début, recherche du sens du navire par un second tir
    while Tour == False: #Prend en compte le fait qu'une des 4 cases potentielles peut avoir été déjà touché
        NewCible = randint(1, 4) #choix aléatoire: Où attaquer maintenant ?
        print("NewCible = " + str(NewCible))
        print("IndexCible = " +str(IndexCible))
        if NewCible == 1:
            if list(Grille1[IndexCible][0])[0]+1 <= 485: #Il y a-t-il une case à droite ?
                CaseCible = Grille1[IndexCible+1] #case à droite
                if CaseCible[2] == 0: #La case a-t-elle été touché ?
                    Grille1[IndexCible+1][2] = 1 #On tir sur la case
                    print("IndexCase = " + str(IndexCible+1))
                    print("Case: " + str(CaseCible))
                    print("Bateau ?: " + str(CaseCible[1]))
                    Code = Dico_IA[CaseCible[0]] #Recherche du nom de la case à partir de ses coordonnées grâce au Dico "CaseCible"
                    print(Code) #Affichage de la case dans la console
                    TextCase = font.render(Code, 1, (0,0,0))
                    fenetre.blit(TextCase, (115, 530)) #Affichage de la case sur la fenêtre de jeu
                    if CaseCible[1] == 1: #Il y a-t-il un bateau sur cette case ?
                        #On affiche touché et on place un carrée rouge
                        print("TOUCHE")
                        fenetre.blit(CroixRouge, CaseCible[0]) #Placement croix
                        fenetre.blit(TextTOUCHE, (258, 530)) #Affichage du texte "Touché" sur le plateau
                        a = random.randint(1, 2) #Générateur de probabilité
                        if a == 1 : #L'IA dit-il un mot (phrase affiché en haut du plateau). 1 == oui, sinon ce sera à un autre tour
                            TexteIAImpactF(fenetre, font) #appel de la fonction
                        droite = True #Le bateau continue bien sur la droite (utile pour le tour suivant)
                        NbAttaque = NbAttaque +1 #On a touché une case de plus du navire
                        IndexCible = IndexCible+1 #on passe à la case suivante (pour le prochain tour)
                        Tour = True #fin du tour de jeu
                    else:
                        #On affiche dans l'eau et on place un carrée bleu
                        print("DANS L'EAU")
                        fenetre.blit(RondBleu, CaseCible[0])  #Placement rond
                        fenetre.blit(TextEAU, (258, 530)) #Affichage du texte "Dans l'eau" sur le plateau
                        a = random.randint(1, 3) #Générateur de probabilité
                        if a == 1 : #L'IA dit-il un mot (phrase affiché en haut du plateau). 1 == oui, sinon ce sera à un autre tour
                            TexteIAEauF(fenetre, font) #appel de la fonction
                        Tour = True #Fin du tour de jeu
                        break #On sort de la boucle while (plus rapide que de laisser le programme en sortir seul)
        elif NewCible == 2:
            if list(Grille1[IndexCible][0])[0]-1 >= 105: #Il y a-t-il une case à gauche ?
                CaseCible = Grille1[IndexCible-1] #case à gauche
```

```
436             #Appelle fonction Nombre de Case
437             CountNbCoulIANavire(Grille1, IndexCible) #comptage du nombre de case du bateau touché (sert pour le coulage)
438             if NbAttaque == NbCoulIANavire: #Le bateau est-til coulé ? (à ce stade ne vaut que pour les navires de 2 cases)
439                 InAttaque = False #On sort de la phase d'attaque pour retourner dans celle de recherche
440                 print("COULE")
441                 fenetre.blit(TextCOULE, (414, 530)) #Affichage du texte "Coulé" sur le plateau
442                 Fonction_Ckeck_Coulage_Fin() #Réalisation de la fonction finale checkant les cases adjacentes au navire pour ne pas les retoucher
443                 NbCoulIA = NbCoulIA+1 #+1 navire coulé, la victoire est proche...
444     else:
445         #Appelle fonction suite d'attaque, lorsque le sens du bateau est trouvé
446         Attaque_Avancee_IA(Tour, Grille1, fenetre, CroixRouge, RondBleu, TextTOUCHE, TextEAU, TextCOULE, Dico_IA, font)
447     print("NbAttaque in End Tour = " + str(NbAttaque))
```

Annexe N°5 : Exemple d'Attaque programme de coulage Phase II

```
while Tour == False: #tourne tant que le tour de l'IA n'est pas terminé (tant qu'il n'a pas officiellement touché une case)
    if droite == True: #Le bateau continue-t-il à droite ?
        if list(Grille1[IndexCible][0])[0]+20 <= 485: #Il y a-t-il une case à droite ?
            CaseCible =  Grille1[IndexCible+1] #on touche prend l'indice suivant
            if CaseCible[2] == 0: #La case a-t-elle déjà été touché ?
                Grille1[IndexCible+1][2] = 1 #On touche la case
                print("IndexCase = " + str(IndexCible+1))
                print("Case: " + str(CaseCible))
                print("Bateau ?: " + str(CaseCible[1]))
                #Affichage de la case touché sur la console et le plateau
                Code = Dico_IA[CaseCible[0]]
                print(Code)
                TextCase = font.render(Code, 1, (0,0,0))
                fenetre.blit(TextCase, (115, 530))
                if CaseCible[1] == 1: #Il y a-t-il un bateau ?
                    #On affiche touché et on place un carrée rouge
                    print("TOUCHE")
                    fenetre.blit(CroixRouge, CaseCible[0]) #Placement croix
                    fenetre.blit(TextTOUCHE, (258, 530)) #Affichage du texte "Touché" sur le plateau
                    a = random.randint(1, 2) #générateur de probabilité
                    if a == 1 : #L'IA dit-il un mot ?
                        TexteIAImpactF(fenetre, font) #appel de la fonction
                    NbAttaque = NbAttaque +1 #On a touché une case supplémentaire
                    IndexCible = IndexCible +1 #cible prochain tour
                    Tour = True #Fin du tour de jeu de l'IA
                else:
                    #On affiche dans l'eau et on place un carrée bleu
                    print("DANS L'EAU")
                    fenetre.blit(RondBleu, CaseCible[0])  #Placement rond
                    fenetre.blit(TextEAU, (258, 530)) #Affichage du texte "Dans l'eau" sur le plateau
                    #Plus de bateau à droite: on continue à gauche pour le prochain tour
                    a = random.randint(1, 3) #générateur de probabilité
                    if a == 1 : #L'IA dit-il un mot ?
                        TexteIAEauF(fenetre, font) #appel de la fonction
                    droite = False #plus de bateau à droite...
                    gauche = True #...donc on continue à gauche
                    IndexCible = FirstCase #On revient à la première case
                    Tour = True #Fin du tour de jeu de l'IA
            else: #Déjà touché ? Alors c'était dans l'eau
                IndexCible = FirstCase #on revient à la première case choisi (pour changer de sens au prochain tour)
                droite = False
                gauche = True
        else: #Plus de case à droite: le bateau continue donc à gauche
            droite = False
            gauche = True
            IndexCible = FirstCase #On revient à la première case
    elif gauche == True: #Le bateau continue-t-il à gauche ?
        if list(Grille1[IndexCible][0])[0]-20 >= 105: #Il y a-t-il une case à gauche ?
```

Annexe N°6 : Programme de Coulage Phase IV (exemple pour un navire Horizontal)

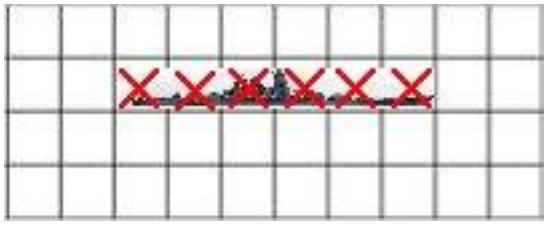
```
744
745 def Fonction_Ckeck_Coulage_Fin():
746     #On globalise les variables
747     global Grille1
748     global bas
749     global haut
750     global gauche
751     global droite
752     global NbCoulIANavire
753     global Reference
754     j = Grille1.index(Reference) #J permet de manipuler IndexCase sans le modifier lui même
755     print("DEBUT CHECK COULAGE DU NAVIRE")
756     print("IndexCase = " + str(j))
757     print("1er case: " + str(Reference))
758     #Variable Bolléenne qui détermine si il existe des cases en dessous/dessus/droite/gauche
759     #de la case choisi (on part du principe que "oui"), True = la (les) case(s) existe(nt)
760     #Variable qui détermine le sens du navire (Horizontale/Verticale)
761     TupleCoord = list(Grille1[j][0])
762     case_dessus = True
763     case_dessous = True
764     case_gauche = True
765     case_droite = True
766     #détermination des cases existante (quatre test)
767     #Si les test sont "Vrais", alors les cases en question n'existent pas
768     if TupleCoord[0]-20 < 105: #Y a-t-il une case à gauche ?
769         case_gauche = False
770     if TupleCoord[0]+20 > 485: #Y a-t-il une case à droite ?
771         case_droite = False
772     if TupleCoord[1]+20 > 485: #Y a-t-il une case en dessous ?
773         case_dessous = False
774     if TupleCoord[1]-20 < 105: #Y a-t-il une case au dessus ?
775         case_dessus = False
```

```
    #Travail sur les cases du dessus (si elles existent):
    if case_dessus == True: #Cases supérieures
        print("passage --> case dessus")
        Grille1[j-20][2] = 1 #On touche la case
        if case_droite == True: #Case supérieure droite (diagonale)
            print("passage --> case dessus droite")
            Grille1[j-19][2] = 1 #On touche la case
        if case_gauche == True: #Case  supérieure gauche (diagonale)
            print("passage --> case gauche")
            Grille1[j-21][2] = 1 #On touche la case

    #Travail sur les cases du dessous (si elles existent):
    if case_dessous == True:
        print("passage --> case dessous")
        Grille1[j+20][2] = 1  #On touche la case
        if case_droite == True: #Case inférieure droite (diagonale)
            print("passage --> case dessous droite")
            Grille1[j+21][2] = 1 #On touche la case
        if case_gauche == True: #Case inférieure gauche (diagonale)
            print("passage --> case dessous gauche")
            Grille1[j+19][2] = 1 #On touche la case

    #Travail sur la case à gauche (si elle existe):
    if case_gauche == True:
        print("passage --> case gauche")
        Grille1[j-1][2] = 1 #On touche la case
    #Travail sur la case à droite (si elle existe):
    if case_droite == True:
        print("passage --> case droite")
        Grille1[j+1][2] = 1  #On touche la case

    #Détermination du type de navire (Horizontale/Verticale) à l'aide des variable H/V:
    if droite == True or gauche == True: #Le bateau est-il Horizontal ?
        #La technique utilisé pourra être réutiliser pour la programme "Coulage des Navires" de l'IA
        print("Check_Coulage_NavireH")
        Check_Coulage_NavireH(j, case_dessus, case_dessous)
    elif bas == True or haut == True: #Le bateau est-il Verticale ?
        #La technique utilisé pourra être réutiliser pour la programme "Coulage des Navires" de l'IA
        print("Check_Coulage_NavireV")
        Check_Coulage_NavireV(j, case_gauche, case_droite)
    print("FIN CHECK COULAGE DU NAVIRE") #Utile à savoir
```

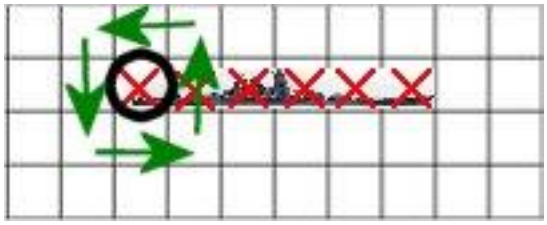
```
def Check_Coulage_NavireH(j, case_dessus, case_dessous): #Spécifique aux navires horizontaux
    #on globalise les variables
    global Grille1
    global droite
    global gauche
    global Reference
    global NbCoulIANavire
    x = 1 #on commence le compteur à un (et on retirera 1 au nombre de case du navire restant)
    for x in range (NbCoulIANavire-1): #La boucle tourne tant que l'on est pas arrivé à la fin du bateau
        j = j+1 #case suivante
        print("Case: " + str(Grille1[j]))
        print("Index: " + str(j))
        if case_dessus == True: #On vérifie que la case au dessus existe...
            Grille1[j-20][2] = 1 #On n'oublie pas de checker la Case
        if case_dessous == True:  #On vérifie que la case en dessous existe...
            Grille1[j+20][2] = 1 #On n'oublie pas de checker la Case
    #Partie qui se fait une fois toutes les cases du navires ckeckées (on va sur les celles juste après)
    if list(Grille1[j+1][0])[0] <= 485: #Y a-t-il une case deux cases à droite ?
        j = j+1
        Grille1[j][2] = 1 #On check la Case juste après le bateau
        if case_dessus == True: #On vérifie que la case au dessus existe...
            Grille1[j-20][2] = 1 #On n'oublie pas de checker la Case
        if case_dessous == True:  #On vérifie que la case en dessous existe...
            Grille1[j+20][2] = 1 #On n'oublie pas de checker la Case
```

Annexe N°7 : Coulage Phase IV, images explicatives

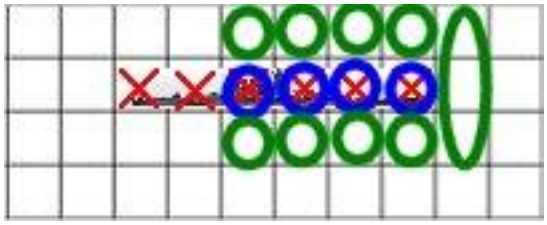
Le programme de checkage des cases fonctionne de la sorte. Tout d'abord, le navire est coulé :



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 terminer 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.