Laboratoire de méthode de programmation

Talbot
Pierre
Groupe 2

Combat Naval

Professeur : B. Naisse

1ère baccalauréat en informatique de gestion 2009/2010

Records

```
var SIZE_O : integer
SIZE_O ⟵ 10

type square : record
        etat : integer
        destroy : boolean
        AI_aura : boolean
        end

type tocean : array[1..SIZE_O,1..SIZE_O] of square

type boat : record
        name : string
        len : integer
        lenRemain : integer
        end

type tflotte : array[1..5] of boat

type profil : record
        ocean : tocean
        boatRemain : integer
        flotte : tflotte
        end

type IA_module : record
        boatFound : boolean
        xi : integer
        yi : integer
        end



var player : profil
var computer : profil
var AI_module : attack_module
```

Commentaire sur la constante

I. SIZE_O

Constante définissant la dimension d'un coté de l'océan. Elle doit être assez grande pour placer tout les bateaux correctement. Il faut faire attention à l'affichage lorsque celle-ci augmente.

Commentaires Records

I. Record square

Définition : Représente une case de l'océan.
Champs :

- **etat** : Représente le contenu d'une case selon les indicateurs suivants :
  - -1 : L'aura[1] d'un bateau.
  - 0 : Eau libre.
  - 1 : Destroyer.
  - 2 : Sous-marin.
  - 3 : Croiseur.
  - 4 : Cuirassé.
  - 5 : Porte-avion.

  D'une manière générale, les indicateurs de bateaux sont > 0 et les autres états d'une case sont <= 0.
- **destroy** et **AI_aura** sont complémentaire dans le sens où :
  - **destroy = TRUE** et **AI_aura = TRUE** : Case torpillée (etat < 1).
  - **destroy = TRUE** et **AI_aura = FALSE** : Case torpillée (etat >= 1).
  - **destroy = FALSE** et **AI_aura = TRUE** : Case non-torpillée et qui ne le sera jamais, l'ordinateur sait que c'est une aura (etat = -1).
  - **destroy = TRUE** et **AI_aura = TRUE** : Case non-torpillée.

II. Record boat

Définition : Représente un bateau.
Champs :

- **name** : Nom du bateau.
- **len** : Longueur initiale du bateau.
- **lenRemain** : Longueur restante du bateau.

[1] Aura : Contour d'un navire sur lequel aucun autre navire ne peut se trouver.

III. Record profil

Définition : Représente un profil de joueur (réel ou virtuel).
Champs :

- **ocean** : Océan du joueur de SIZE_O * SIZE_O cases.
- **boatRemain** : Bateaux restants du joueur.
- **flotte** : Flotte navale du joueur comprenant 5 bateaux.

IV. Record IA_module

Définition : Aide aux choix du joueur virtuel sur les cases a torpillées.
Champs :

- **boatFound** : TRUE si l'ordinateur a trouvé un bateau mais ne l'a pas encore coulé.
- **xi** : Coordonnée des abscisses de la dernière case torpillée qui a touché un bateau.
- **yi** : Coordonnée des ordonnées de la dernière case torpillée qui a touché un bateau.

Commentaires sur les types

I. tocean

Résultat d'une déclaration : tableau de type square en 2 dimensions ([y,x]) de taille SIZE_O où y représente les lignes et x les colonnes.

II. tflotte

Résultat d'une déclaration : tableau de type boat de taille 5 (nombre de bateau total).

Commentaires sur les variables principales

var player : profil

Nous avons ici notre variable du joueur réel ‘modélisant’ son profil comme indiqué dans le record profil.

var computer : profil

Nous avons ici notre variable du joueur virtuel ‘modélisant’ son profil comme indiqué dans le record profil.

var AI_module : attack_module

C’est la variable d’aide aux choix de l’ordinateur sur les cases a torpillées. Elle n’est donc utile que dans la partie d’attaque au tour à tour.

Commentaires sur les différentes notations utilisées dans ce projet

Nomenclature

t : utilisé pour dire type. Exemple : tocean.
p : utilisé pour dire player. Exemple : pOcean.
c : utilisé pour dire computer. Exemple : cOcean.

Profondeur

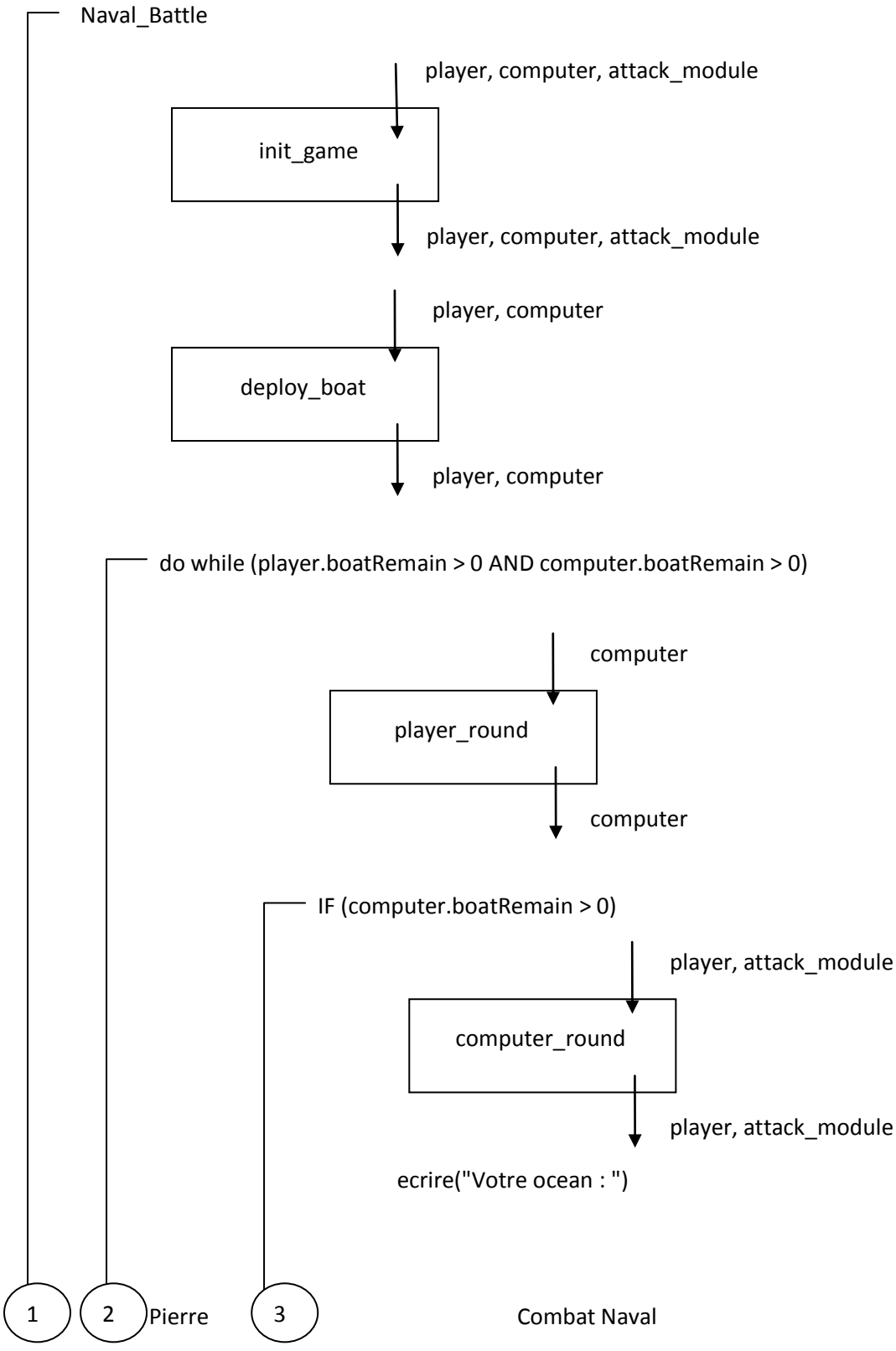
Le terme « profondeur » sera utilisé pour décrire à quel niveau de « profondeur » dans l’imbrication des algorithmes nous sommes. Par exemple :
Sous-programme de : Naval_Battle (Profondeur de niveau 1, niveau actuel : 2) nous informe que la procédure/fonction est invoquée depuis la partie Naval_Battle qui est le programme premier. Nous nous trouvons donc au second « étage » dans notre imbrication.

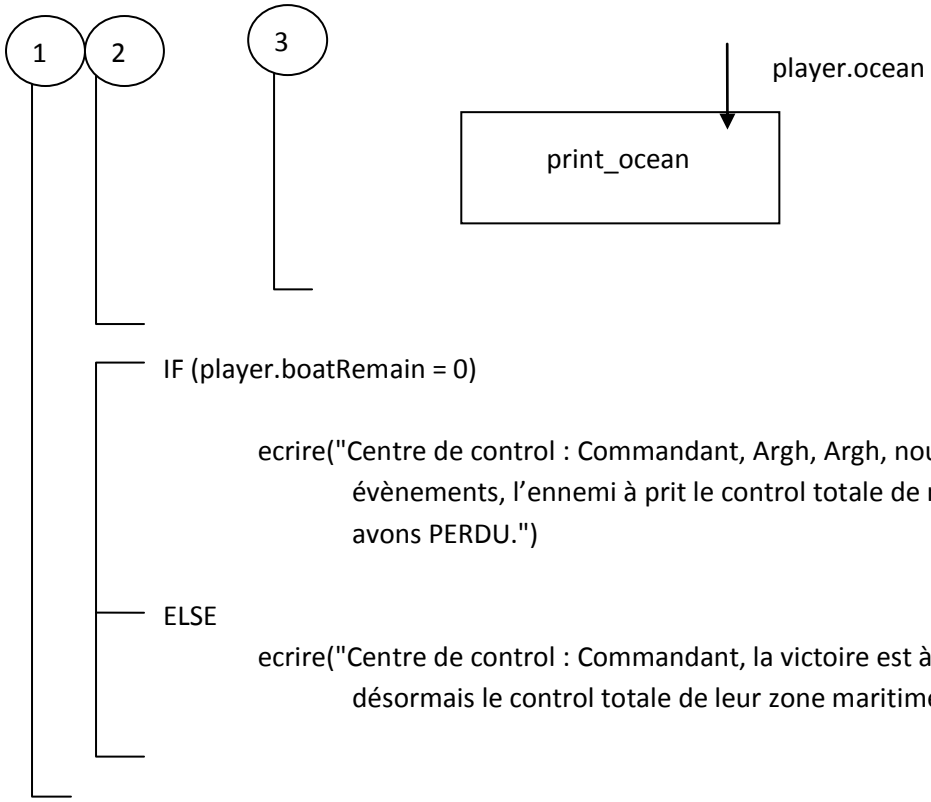
Commentaires de procédures/fonctions

Certaines fonctions ou procédures sont gratifiées par un commentaire, tous les commentaires se baseront sur le fait qu’on suppose être dans un océan de taille 10 sur 10. Cela nous permettra parfois d’expliquer notre algorithme avec des exemples concrets.

Pseudo-code et spécifications : Bataille navale

NAVAL_BATTLE





IF (player.boatRemain = 0)

ecrire("Centre de control : Commandant, Argh, Argh, nous sommes dépassé par les évènements, l'ennemi à prit le control totale de notre zone maritime. Nous avons PERDU.")

ELSE

ecrire("Centre de control : Commandant, la victoire est à nous ! Nous possédons désormais le control totale de leur zone maritime. Nous avons GAGNE.")

INIT_GAME

Sous-programme de : Naval_Battle (Profondeur de niveau 1, niveau actuel : 2).

Paramètres en entrée/sortie : player, computer, attack_module

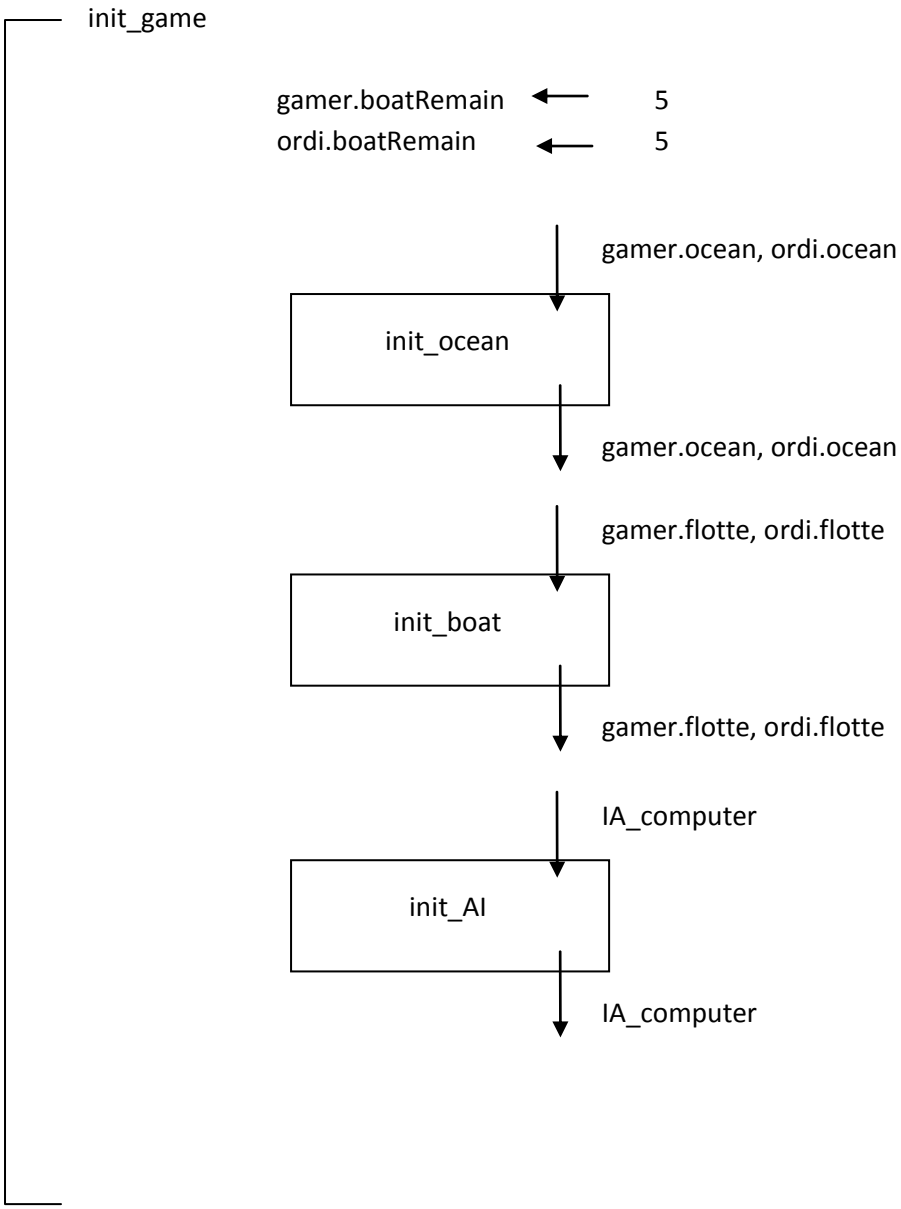
Entête : init_game(var gamer : profil, var ordi : profil, var IA_computer : IA_module)

Pré-conditions : gamer, ordi et IA_computer sont déclarés.

Post-conditions :

- gamer et ordi sont complètements initialisés de la même façon :
  I. Le champ boatRemain est initialisé à 5 (nombre maximum de bateau restant).
  I. Le champ ocean : toutes les cases d'ocean sont complètements initialisées[2].
  II. Le champ flotte : toutes les cases de flotte sont complètement initialisées[3].
- AI_computer est initialisés[4].

[2] cf. spécifications de init_ocean pour les détails de l'initialisation des champs de ocean.
[3] cf. spécifications de init_boat pour les détails de l'initialisation des champs de flotte.
[4] cf. spécifications de init_AI pour les détails de l'initialisation des champs de AI_computer.


init_game
gamer.boatRemain ← 5
ordi.boatRemain ← 5
gamer.ocean, ordi.ocean
init_ocean
gamer.ocean, ordi.ocean
gamer.flotte, ordi.flotte
init_boat
gamer.flotte, ordi.flotte
IA_computer
init_AI
IA_computer

INIT_OCEAN

Sous-programme de : init_game (Profondeur de niveau 2, niveau actuel : 3).

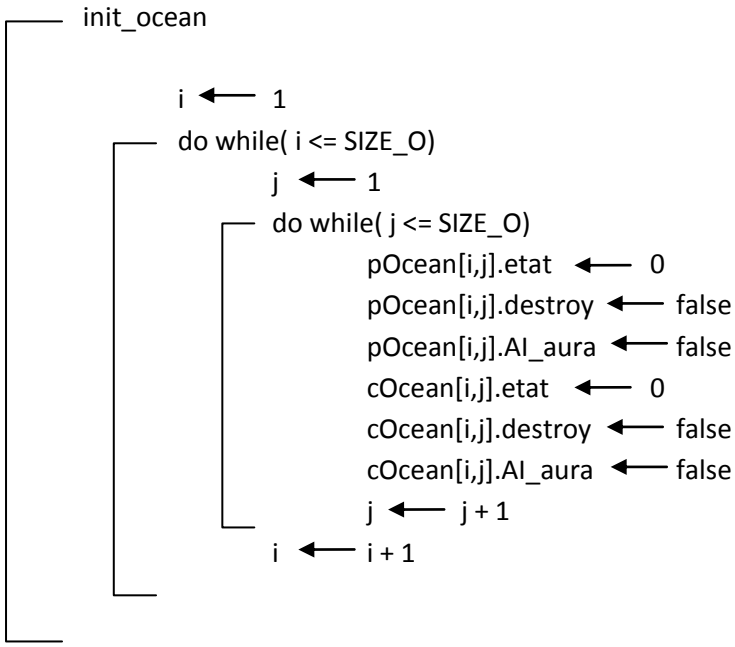
Paramètres en entrée/sortie : gamer.ocean, ordi.ocean

Entête : init_game(var pOcean : tocean, var cOcean : tocean)

Pré-conditions : pOcean et cOcean sont déclarés.

Post-conditions : pOcean et cOcean sont complètements initialisés, pour tout i de j (0<i<=SIZE_O et 0<j <=SIZE_O) dans [i,j] :

I. Le champ etat est initialisé à 0.
II. Les champs destroy et AI_aura sont initialisés à false.



INIT_BOAT

Sous-programme de : init_game (Profondeur de niveau 2, niveau actuel : 3).

Paramètres en entrée/sortie : gamer.flotte, ordi.flotte

Entête : init_game(var pFlotte : tflotte, var cFlotte : tflotte)

Pré-conditions : pFlotte et cFlotte sont déclarés.

Post-conditions : pFlotte et cFlotte sont complètements initialisés, pour tout i (0 < i <=5) nous avons :

I. Le champ nom initialisé avec les noms des bateaux selon l'énoncé.
II. Les champs len et lenRemain initialisés avec la longueur totale du bateau en relation avec son nom selon l'énoncé.

Table des variables :

a. *boat_names* : Tableau de 5 cases comportant les noms des 5 bateaux.
b. *boat_len* : Tableau de 5 cases comportant la longueur de 5 bateaux.

```
init_boat
        boat_names[1]  ←  ″destroyer″
        boat_names[2]  ←  ″sous-marin ″
        boat_names[3]  ←  ″croiseur″
        boat_names[4]  ←  ″cuirasse″
        boat_names[5]  ←  ″porte-avion″
        boat_len[1]    ←  2
        boat_len[2]    ←  3
        boat_len[3]    ←  3
        boat_len[4]    ←  4
        boat_len[5]    ←  5
        i  ←  1
        do while(i <=5)
                pFlotte[i].nom          ←  boat_names[i]
                pFlotte[i].len          ←  boat_len[i]
                pFlotte[i].lenRemain    ←  boat_len[i]
                cFlotte[i].nom          ←  boat_names[i]
                cFlotte[i].len          ←  boat_len[i]
                cFlotte[i].lenRemain    ←  boat_len[i]
                i  ←  i + 1
```

INIT_AI

Sous-programme de : init_game (Profondeur de niveau 2, niveau actuel : 3).

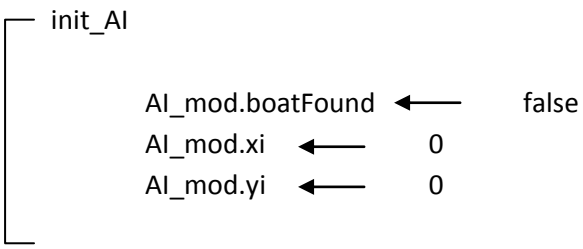
Paramètre en entrée/sortie : attack_module.

Entête : init_AI(var AI_mod : AI_module)

Pré-condition : AI_mod est déclaré.

Post-condition : AI_mod est complètement initialisé :

- Le champ boatFound est initialisé à false.
- Les champs xi et yi sont initialisés à 0.



DEPLOY_BOAT

Sous-programme de : Naval_Battle (Profondeur de niveau 1, niveau actuel : 2).

Paramètres en entrée/sortie : player, computer

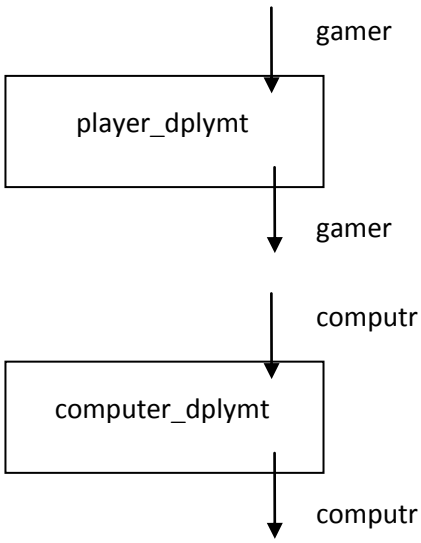
Entête : deploy_boat (var gamer : profil, var computr : profil)

Pré-conditions : gamer et computr sont complètements initialisés selon les spécifications d'init_game.

Post-conditions : gamer et computr sont modifiés, les 5 bateaux sont correctement placé dans leur champ respectif « ocean ». De plus le champ « etat » du champ « ocean » est modifié en conséquence selon les descriptions du record ocean.

deploy_boat

```
ecrire("Mission 1 : Deploiement des bateaux.")
ecrire("-----------------------------------------------")
```



PLAYER_DPLYMT

Sous-programme de : deploy_boat (Profondeur de niveau 2, niveau actuel : 3).

Paramètres en entrée/sortie : gamer

Entête : deploy_boat(var playr : profil)

Pré-condition : playr est complètement initialisé.

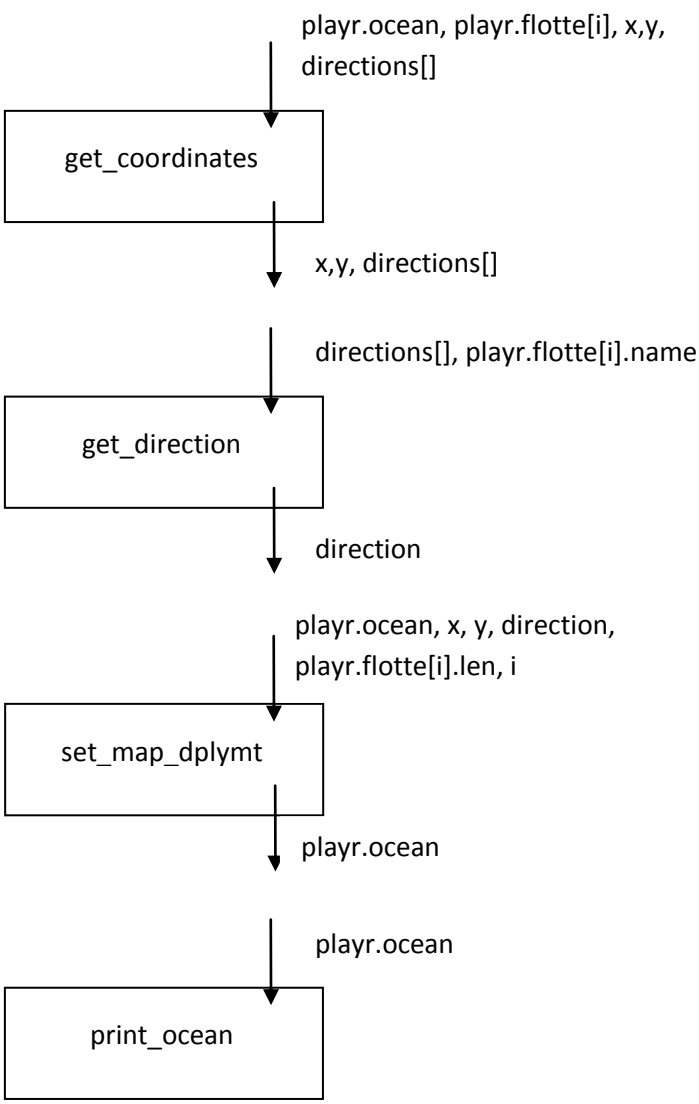
Post-condition : le champ « ocean » de playr est modifié selon la post-condition de set_map_dplymt.

player_dplymt

i ← 1

do while (i <= 5)

ecrire("Caracteristiques du bateau : ")
ecrire("Nom : ",playr.flotte[i].nom)
ecrire("Longueur : ", playr.flotte[i].len)



i ← i + 1

PRINT_OCEAN

Sous-programme de : player_dplymt (Profondeur de niveau 3, niveau actuel : 4).

Paramètre en entrée : playr.ocean

Entête : print_ocean(var oceanToPrint : tocean)

Pré-condition : oceanToPrint est complètement initialisé.

Post-condition : oceanToPrint est inchangé.

Table des variables :

a. caracter : Contient les initiales françaises des bateaux sauf pour le cuirassé : B pour Battleship.

Commentaire :
La première boucle affiche la première ligne : les coordonnées sur l'axe des abscisses (de 1 à 10). Ensuite les deux autres boucles imbriquées affichent la matrice. Elle commence par une lettre (de A à J) et affiche le contenu (soit le champ « etat ») des cases selon la légende suivante :

- Torpillé mais à l'eau : °
- Bateau touché : *
- Cases non torpillées : . (point)
- Bateau non torpillé : initial du bateau en français sauf pour Cuirassé qui devient Battleship avec comme initiale B.

```
print_ocean

        caracter[1] ◄—— ‘D’
        caracter[2] ◄—— ‘S’
        caracter[3] ◄—— ‘C’
        caracter[4] ◄—— ‘B’
        caracter[5] ◄—— ‘P’
        i ◄—— 1
    do while(i <= SIZE_O)
            ecrire (i)
            i ◄—— i + 1
        i ◄—— 1
    do while( i<= SIZE_O)
            ecrire(64+i)
            j ◄—— 1
        do while( j <= SIZE_O)
            IF (NOT oceanToPrint[i,j].destroy)
                IF (oceanToPrint[i,j].etat <= 0)
                        ecrire(‘.’)
                ELSE
                        ecrire(caracter[oceanToPrint[i,j].etat])
            ELSE
                IF(oceanToPrint[i,j].etat <= 0)
                        ecrire(‘°’)
                ELSE
                        ecrire(‘*’)
                j ◄—— j + 1
        i◄—— i + 1

        ecrire("Légende : ")
        ecrire("Torpillé mais à l’eau : ° ")
        ecrire("Bateau touché : * ")
        ecrire("Cases non torpillées : . ")
        ecrire("Bateau non torpillées : initial du bateau en français sauf pour Cuirassé qui devient
                Battleship avec comme initiale B")
```

GET_COORDINATES

Sous-programme de : player_dplymt (Profondeur de niveau 3, niveau actuel : 4).

Paramètres en entrée : playr.ocean, playr.flotte[i]

Paramètres en entrée/sortie : x,y, directions[]

Entête : get_coordinates(var pOcean : tocean, var bateau : boat, var xa : integer, var ya : integer, var dir : array[1..4] of boolean)

Pré-conditions :

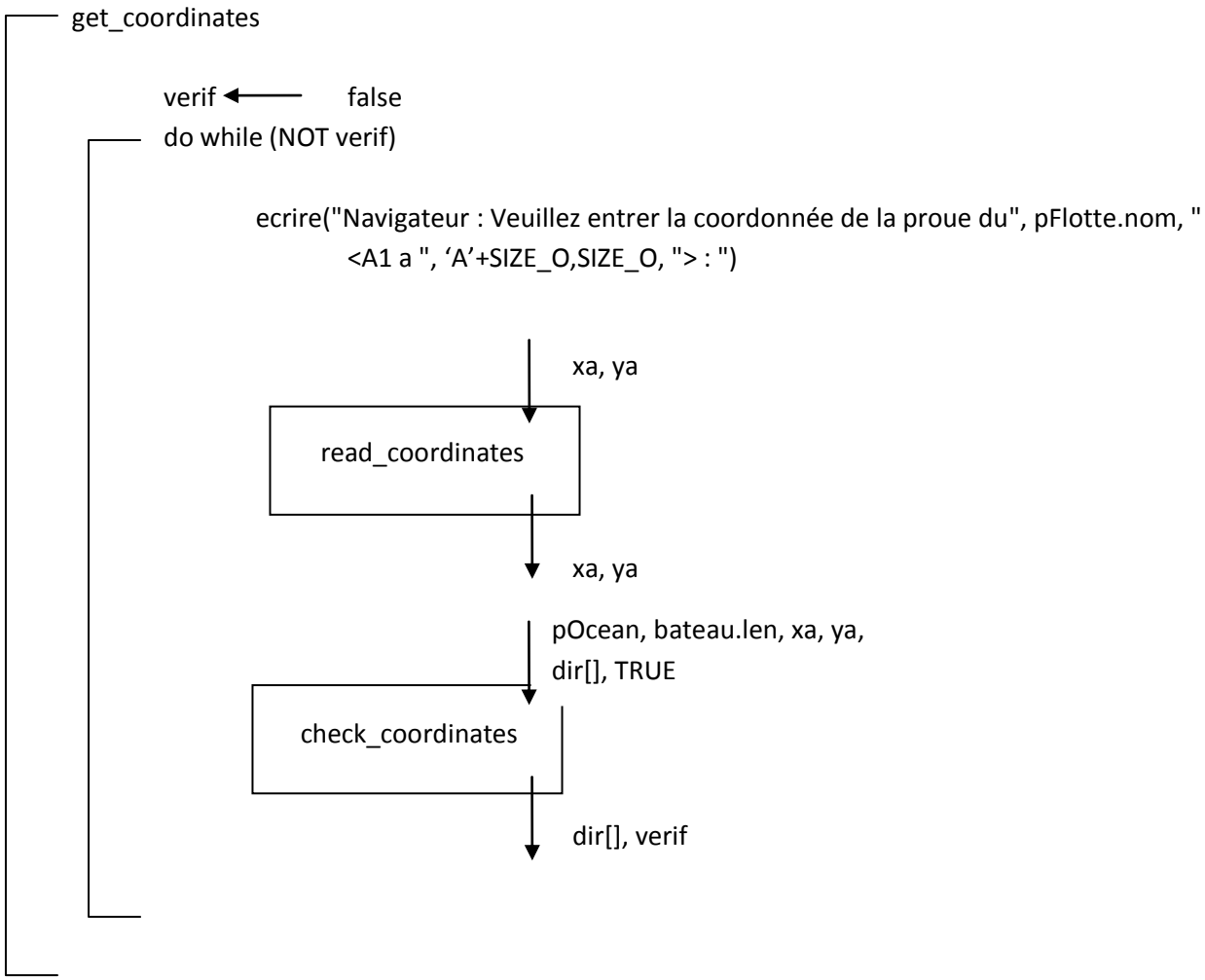
- pOcean et bateau sont complètements initialisés.
- xa, ya et dir sont déclarés.

Post-conditions :

- pOcean et bateau sont inchangés.
- xa et ya sont initialisés avec des coordonnées valides selon les critères de vérification de check_coordinates.
- dir est initialisé selon la post-condition de check_coordinates.

Table des variables :

a. verif : Booléen permettant de sortir de la boucle si check_coordinates renvoie TRUE.


get_coordinates
verif ← false
do while (NOT verif)
ecrire("Navigateur : Veuillez entrer la coordonnée de la proue du", pFlotte.nom, "
<A1 a ", 'A'+SIZE_O,SIZE_O, "> : ")
xa, ya
read_coordinates
xa, ya
pOcean, bateau.len, xa, ya,
dir[], TRUE
check_coordinates
dir[], verif

READ_COORDINATES

Sous-programme de : get_coordinates (Profondeur de niveau 4, niveau actuel : 5).

Paramètres en entrée/sortie: xa,ya

Sous-programme de player_round (Profondeur de niveau 2, niveau actuel : 3).

Paramètres en entrée/sortie: x,y

Entête : get_coordinates(var xb : integer, var yb : integer).

Pré-conditions : xb et yb sont déclarés.

Post-conditions : xb et yb sont initialisés, non vérifiés et contiennent les coordonnées encodées par l'utilisateur.

Table des variables :

a. coord : Contient l'entrée de l'utilisateur préalablement initialisé à '.'. Voir exemples.

Commentaire :
Code ASCII de 0 : 48
Code ASCII de @ : 64

Exemples :

- Dans un océan de 10 sur 10. Si l'utilisateur entre A5 :

Visualisation des codes ASCII en mémoire dans le tableau coord :

65	53	46

65 : A 53 : 5 46 : .

xb ⟵ 53 – 48

yb ⟵ 65 – 64

Résultat : x = 5 et y = 1. OK.

- Dans un océan de 25 sur 25. Si l'utilisateur entre M22 :

Visualisation des codes ASCII en mémoire dans le tableau coord :

77	50	50

77 : M 50 : 2 50 : 2

xb ⟵ 50 – 48

yb ⟵ 77 – 64

Résultat : x = 2 et y = 13. La coordonnée y est OK mais pas x.

Lorsqu'il y a un chiffre dans la dernière case on exécute cette dernière instruction :

xb ⟵ xb + (50-48)*10

xb ⟵ 2 + 20

Résultat : x = 22. OK.

```
read_coordinates

        i ⟵ 1
        do while (i<=3)
                coord[i] ⟵ '.'
                i ⟵ i + 1
        lire(coord)
        xb ⟵ coord[2] – '0'
        yb ⟵ coord[1] – '@'
        IF (coord[3] >= '0' AND coord[3] <= '9')
                xb ⟵ xb + (coord[3] – 48) * 10
```

CHECK_COORDINATES

Sous-programme de : get_coordinates(Profondeur de niveau 4, niveau actuel : 5).
Paramètres en entrée : pOcean, pFlotte[i].len, xa, ya, TRUE

Sous-programme de : getIA_coordinates(Profondeur de niveau 4, niveau actuel : 5).
Paramètres en entrée : iOcean, length, xi, yi, FALSE

Paramètre en sortie : verif

Paramètre en entrée/sortie : dir[]

Entête : check_coordinates(var oceanToCheck : tocean, length : integer, xb : integer, yb : integer, var direction : array[1..4] of Boolean, msg : boolean) check : boolean

Pré-conditions :

- oceanToCheck est complètement initialisé.
- length est initialisé et contient la longueur du bateau qu'on veut placer.
- xb et yb sont initialisés, non vérifiés et contiennent les coordonnées encodées par l'utilisateur.
- msg est initialisé à TRUE si on veut afficher les messages utilisateurs, à FALSE sinon.

Post-conditions :

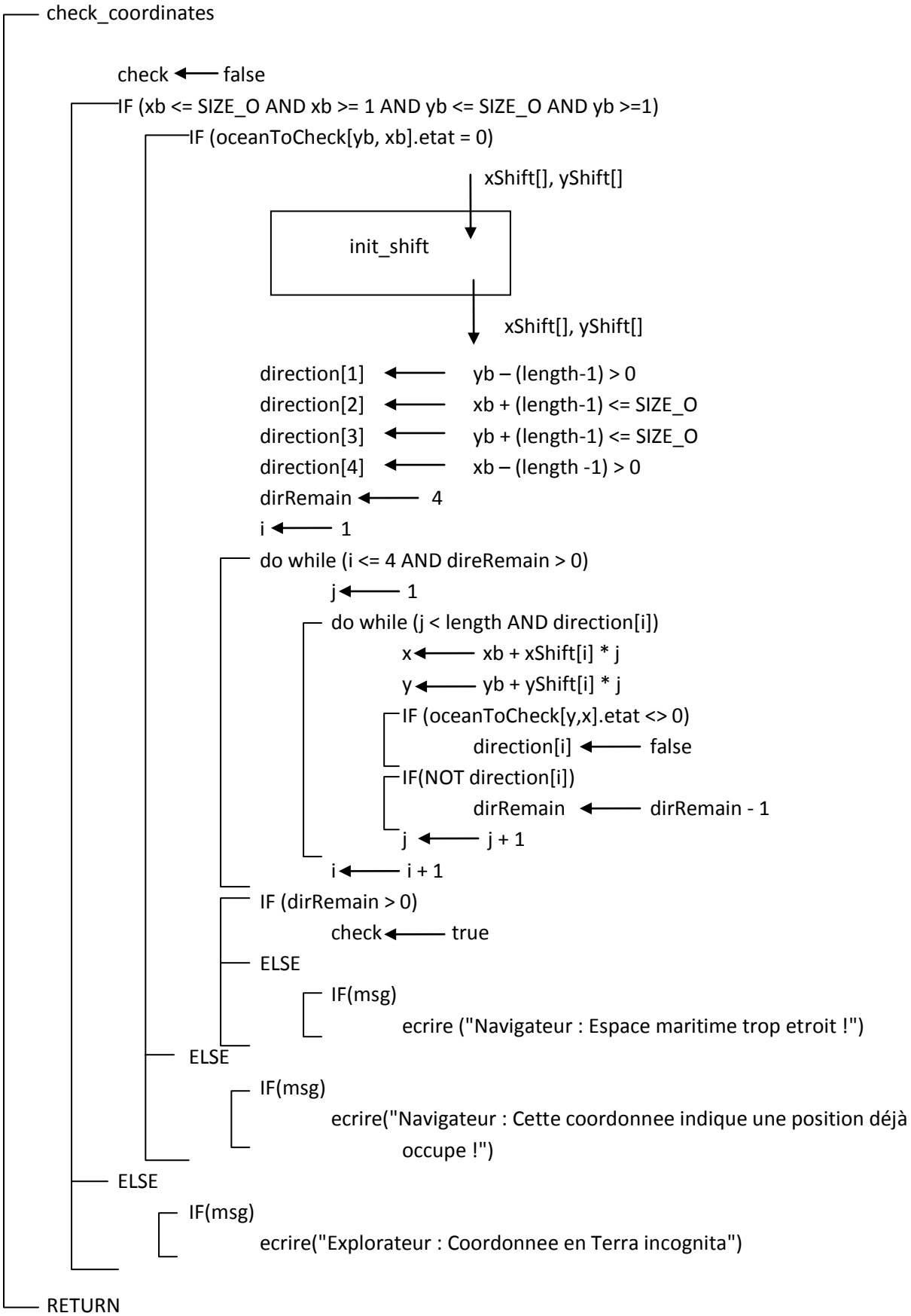
- oceanToCheck, length, xb, yb et msg sont inchangés.
- direction est initialisée avec quatre booléens représentants les autorisations de placer le bateau dans les quatre directions polaires. Dans l'ordre : le nord, l'est, le sud et l'ouest. Pour tout i entre 1 et 4 compris, dir[i] = TRUE si le bateau peut-être positionné dans cette direction, sinon FALSE.

Résultat : TRUE si les coordonnées xb et yb sont valides, FALSE sinon. Une coordonnée valide se trouve entre [1,1] et [SIZE_O,SIZE_O] et dans une case libre, il faut également assez de place (length cases) pour placer le bateau dans, minimum, 1 direction polaire.

Table des variables :

a. dirRemain : Le nombre de directions polaires restantes initialisé à 4.

Commentaire :
Que font les deux boucles imbriquées ? La première décale dans chacune des 4 directions polaires (voir commentaire init_shift), la deuxième se charge de décaler une seconde fois dans la direction voulue sur length cases, si une case n'est pas libre ou alors que nous sortons de l'océan (initialisation de direction en dehors des boucles), nous n'autorisons pas le placement du bateau dans cette direction.
Pourquoi ce test ne se trouve pas plus logiquement dans le sous-programme get_direction ? Il fallait vérifier que notre bateau pouvait être positionné dans au moins une direction, dès lors nous pouvions aussi déterminer dans quelles directions il pouvait être placé.
La variable booléenne message nous permet de réutiliser cette boite dans deux cas de figures, nous n'allions pas écrire deux fois la même chose juste pour une histoire d'affichage de message.


check_coordinates
check ← false
IF (xb <= SIZE_O AND xb >= 1 AND yb <= SIZE_O AND yb >=1)
IF (oceanToCheck[yb, xb].etat = 0)
xShift[], yShift[]
init_shift
xShift[], yShift[]
direction[1] ← yb – (length-1) > 0
direction[2] ← xb + (length-1) <= SIZE_O
direction[3] ← yb + (length-1) <= SIZE_O
direction[4] ← xb – (length -1) > 0
dirRemain ← 4
i ← 1
do while (i <= 4 AND direRemain > 0)
j ← 1
do while (j < length AND direction[i])
x ← xb + xShift[i] * j
y ← yb + yShift[i] * j
IF (oceanToCheck[y,x].etat <> 0)
direction[i] ← false
IF(NOT direction[i])
dirRemain ← dirRemain - 1
j ← j + 1
i ← i + 1
IF (dirRemain > 0)
check ← true
ELSE
IF(msg)
ecrire ("Navigateur : Espace maritime trop etroit !")
ELSE
IF(msg)
ecrire("Navigateur : Cette coordonnee indique une position déjà
occupe !")
ELSE
IF(msg)
ecrire("Explorateur : Coordonnee en Terra incognita")
RETURN

INIT_SHIFT

Sous-programme de : check_coordinates (Profondeur de niveau 5, niveau actuel : 6).

Sous-programme de : set_map_dplymt (Profondeur de niveau 4, niveau actuel : 5).

Paramètre en entrée/sortie : xShift[], yShift[]

Entête : init_shift(var shiftX : array[1..4] of integer, var shiftY : array[1..4] of integer)

Pré-conditions : shiftX et shiftY sont déclarés.

Post-conditions : shiftX et shiftY sont initialisés à des valeurs spécifiques permettant d'effectuer 4 décalages sur une coordonnée. Pour tout i entre 1 et 4 et pour un décalage sur [y,x] : [y * shiftY[i], x * shiftX[i]] nous permet d'obtenir la coordonnée jouxtant [y,x] dans la direction i par rapport a [y,x] où i représente 4 directions polaires dans cet ordre : Nord – Est – Sud – Ouest.

Commentaire :

Nous testons nos deux tableaux sur une coordonnée en x = 5 et y = 5. Nous obtenons bien le décalage voulu.

	[yN,xN] : [4,5]		xN = X + shiftX[1] =>xN = 5 yN = Y + shiftY[1] => yN = 4
[yO, xO] : [5,4]	[Y,X] : [5,5]	[yE, xE] : [5,6]	xE = X + shiftX[2] =>xE = 6 yE = Y + shiftY[2] => yE = 5 xO = X + shiftX[4] =>xO = 4 yO = Y + shiftY[4] => yO = 5
	[yS, xS] : [6,5]		xS = X + shiftX[3] =>xS = 5 yS = Y + shiftY[3] => yS = 6

init_shift

shiftX[1] ⟵ 0
shiftX[2] ⟵ 1
shiftX[3] ⟵ 0
shiftX[4] ⟵ -1
shiftY[1] ⟵ -1
shiftY[2] ⟵ 0
shiftY[3] ⟵ 1
shiftY[4] ⟵ 0

GET_DIRECTION

Sous-programme de : player_dplymt (Profondeur de niveau 4, niveau actuel : 5).

Paramètres en entrée : directions[], playr.flotte[i]

Paramètre en sortie : direction

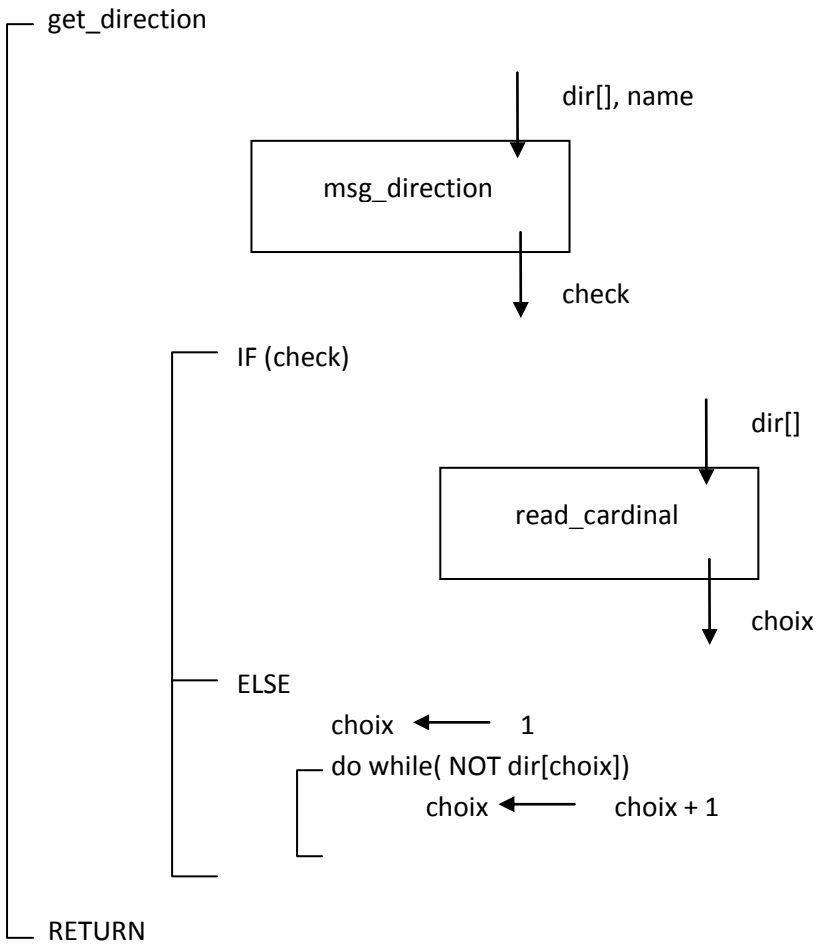
Entête : get_direction(var dir : array[1..4] of boolean, name : string) choix : integer

Pré-conditions :

- dir est complètement initialisé avec les autorisations de direction selon la post-condition de check_coordinates.
- name est initialisé avec le nom du bateau à placer.

Post-conditions : dir et name sont inchangés.

Résultat : choix est initialisé avec la direction choisie par l'utilisateur si les directions restantes sont supérieures à 1. Sinon la direction est choisie automatiquement. Choix contient 1 pour le nord, 2 pour l'est, 3 pour le sud et 4 pour l'ouest.



MSG_DIRECTION

Sous-programme de : get_direction (Profondeur de niveau 5, niveau actuel : 6).

Paramètres en entrée : dir[], name

Paramètre en sortie : check

Entête : msg_direction(var direction : array[1..4] of boolean, nom : string) continuer : boolean

Pré-conditions :

- direction est complètement initialisée avec les autorisations de direction selon la post-condition de check_coordinates.
- nom est initialisé avec le nom du bateau à placer.

Post-conditions : direction et nom sont inchangés.

Résultat : TRUE si l'utilisateur doit encoder son choix, FALSE si la direction a été choisie automatiquement (il n'y avait que un choix).

Table des variables :

a) polaire[] : Les 4 directions polaires en français.
b) dirRemain : Le nombre de direction restante.
c) cpt : Compteur permettant de savoir lorsqu'on arrive à la fin, pour ne pas afficher une virgule à la place d'un point de fin phrase.

Commentaire : Nous proposons en français les directions restantes au joueur en vue d'encoder son choix. S'il ne reste qu'une direction, elle sera choisie automatiquement et aucun encodage ne sera réalisé.

```
msg_direction
        continuer ⟵ true
        polaire[1] ⟵ "au nord <N>"
        polaire[2] ⟵ "a l'est <E>"
        polaire[3] ⟵ "au sud <S>"
        polaire[4] ⟵ "a l'ouest <O>"
        dirRemain ⟵ 4
        i ⟵ 1
    do while( i<= 4)
        IF(NOT direction[i])
            dirRemain ⟵ dirRemain – 1
    IF( dirRemain > 1)
        ecrire ("Tacticien : Il y a ", dirRemain, "directions disponibles : ")
        cpt ⟵ 1
        i ⟵ 1
        do while( i<=4)
            IF (direction[i])
                ecrire(polaire[i])
                IF (cpt <> dirRemain)
                    ecrire(",")
                cpt ⟵ cpt + 1
            i ⟵ i + 1
        ecrire(".Commandant, quelle est votre choix ? ")
    ELSE
        i ⟵ 1
        do while(NOT dir[i])
            i ⟵ i + 1
        ecrire("Tacticien : Le ",  name, " a ete place ", polaire[i])
        continuer ⟵ false
RETURN
```

READ_CARDINAL

Sous-programme de : get_direction (Profondeur de niveau 5, niveau actuel : 6).

Paramètres en entrée : dir[]

Paramètre en sortie : choix

Entête : read_cardinal(direction : array[1..4] of boolean) choice : integer

Pré-condition : direction est complètement initialisé avec les autorisations de direction selon la post-condition de check_coordinates.

Post-condition : direction est inchangé.

Résultat : choice contient une direction choisie par l'utilisateur entre plusieurs disponibles : choice contient 1 pour le nord, 2 pour l'est, 3 pour le sud, 4 pour l'ouest.

```
read_cardinal
        choix[1] ← 'N'
        choix[2] ← 'n'
        choix[3] ← 'E'
        choix[4] ← 'e'
        choix[5] ← 'S'
        choix[6] ← 's'
        choix[7] ← 'O'
        choix[8] ← 'o'
        check ← true
    do while(check)
            lire(choice)
            i ← 2
        do while(i <= 8 AND check)
                IF( dir[i/2] AND (choix[i-1]=choice OR choix[i] = choice))
                        check ← false
                i ← i+2
        IF(check)
                ecrire("Cette direction n'est pas disponible ! Veuillez encoder une des
                        directions proposées.")
    choice ← i/2
RETURN
```

SET_MAP_DPLYMT

Sous-programme de : playr_dplymt(Profondeur de niveau 3, niveau actuel : 4).

Paramètres en entrée : x, y, direction, playr.flotte[i].len, i

Paramètre en entrée/sortie : playr.ocean

Sous-programme de : computer_dplymt(Profondeur de niveau 3, niveau actuel : 4).

Paramètres en entrée : x, y, direction, ordi.flotte[i].len, i

Paramètre en entrée/sortie : ordi.ocean

Entête : set_map_dplymt(var oceanToSet : tocean , xi : integer, yi : integer , dir : integer , length : integer, boat_n : integer)

Pré-conditions : oceanToSet, xi, yi, dir, length, boat_n sont complètements initialisés.

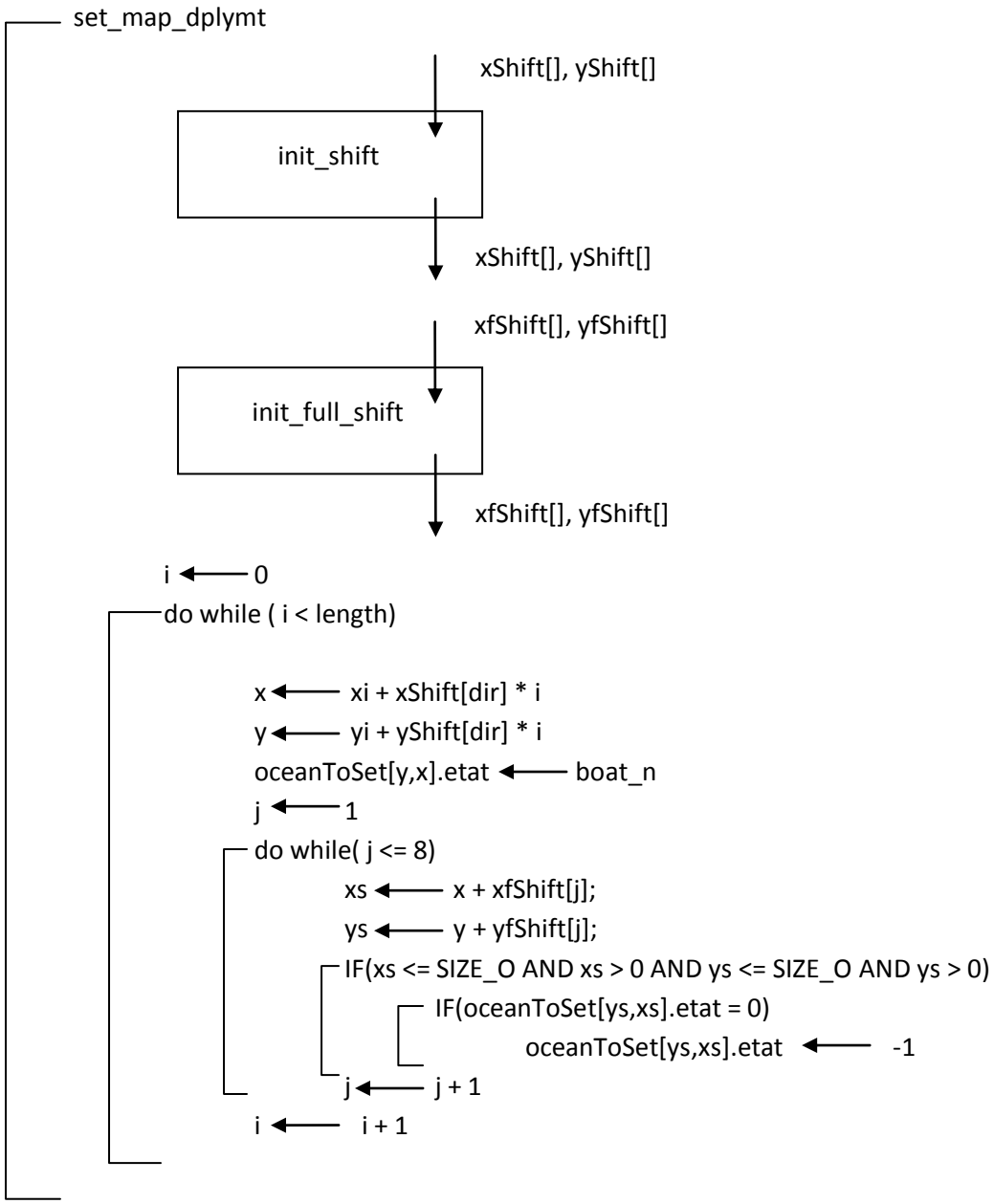
Post-conditions :

- xi, yi, dir, length, boat_n sont inchangés.
- oceanToSet est modifié. L'état des cases où doit être placé le bateau sont modifiés à n_boat. Le bateau est placé à xi, yi sur une longueur length dans la direction dir. Les cases autour du bateau sont modifiées à -1 indiquant l'aura du bateau.

Table des variables :

a) x et y : Ce sont les coordonnées soumises au décalage sur la longueur.
b) xs et ys : Ce sont les coordonnées soumises au décalage autour d'une case, en l'occurrence x et y.

Commentaire :
La première boucle se charge d'attribuer le numéro du bateau (n_boat) sur length cases à partir de la position initiale (xi, yi) à chaque champ etat de ces cases dans la direction polaire dir. En résumé on place le bateau dans l'océan. Le placement du bateau n'est plus vérifié, ceci a été fait antérieurement (cf. check_coordinates). La deuxième boucle imbriquée se charge de placé une « aura » autours de chaque case, nous plaçons ainsi facilement le contour de tout le bateau. Nous utilisons les tableaux de init_full_shift qui permettent de décaler des coordonnées pour tourner tout autour d'une case et celui de init_shift pour avancer dans la bonne direction lors du placement du bateau.


set_map_dplymt
xShift[], yShift[]
init_shift
xShift[], yShift[]
xfShift[], yfShift[]
init_full_shift
xfShift[], yfShift[]
i ← 0
do while (i < length)
x ← xi + xShift[dir] * i
y ← yi + yShift[dir] * i
oceanToSet[y,x].etat ← boat_n
j ← 1
do while(j <= 8)
xs ← x + xfShift[j];
ys ← y + yfShift[j];
IF(xs <= SIZE_O AND xs > 0 AND ys <= SIZE_O AND ys > 0)
IF(oceanToSet[ys,xs].etat = 0)
oceanToSet[ys,xs].etat ← -1
j ← j + 1
i ← i + 1

INIT_FULL_SHIFT

Sous-programme de : set_map_dplymt (Profondeur de niveau 4, niveau actuel : 5).

Sous-programme de : set_aura_border (Profondeur de niveau 4, niveau actuel : 5).

Paramètre en entrée/sortie : xfShift[], yfShift[]

Entête : init_full_shift (var fShiftX : array[1..4] of integer, var fShiftY : array[1..4] of integer)

Pré-conditions : fShiftX et fShiftY sont déclarés.

Post-conditions : fShiftX et fShiftY sont initialisés à des valeurs spécifiques permettant d'effectuer 8 décalages sur une coordonnée. Pour tout i entre 1 et 8 et pour un décalage sur une origine en [y,x], [y * fShiftY[i], x * fShiftX[i]] nous permet d'obtenir la coordonnée jouxtant [y,x] dans la direction i par rapport à [y,x] où i représente 8 directions polaires dans cet ordre : Nord – Nord-Est – Est – Sud-Est – Sud – Sud-Ouest – Ouest – Nord-Ouest.

Commentaire :

Cette procédure est du même gabarit que init_shift, qui rappelons-nous « tournait » autour d'une case en passant par le Nord, Est, Sud et l'Ouest. Celle-ci permet de tourner tout autour d'une case, positions diagonales comprises. Nous aurions pu nous contenter d'un décalage complet (init_full_shift) car init_shift se retrouve dans init_full_shift. Dans une boucle nous aurions du incrémenté i de 2 pour retrouver le décalage de init_shift. Notre solution en deux tableaux me semble néanmoins plus claire.

```
init_full_shift
        fShiftX[1] ⟵ 0
        fShiftX[2] ⟵ 1
        fShiftX[3] ⟵ 1
        fShiftX[4] ⟵ 1
        fShiftX[5] ⟵ 0
        fShiftX[6] ⟵ -1
        fShiftX[7] ⟵ -1
        fShiftX[8] ⟵ -1

        fShiftY[1] ⟵ -1
        fShiftY[2] ⟵ -1
        fShiftY[3] ⟵ 0
        fShiftY[4] ⟵ 1
        fShiftY[5] ⟵ 1
        fShiftY[6] ⟵ 1
        fShiftY[7] ⟵ 0
        fShiftY[8] ⟵ -1
```

COMPUTER_DPLYMT

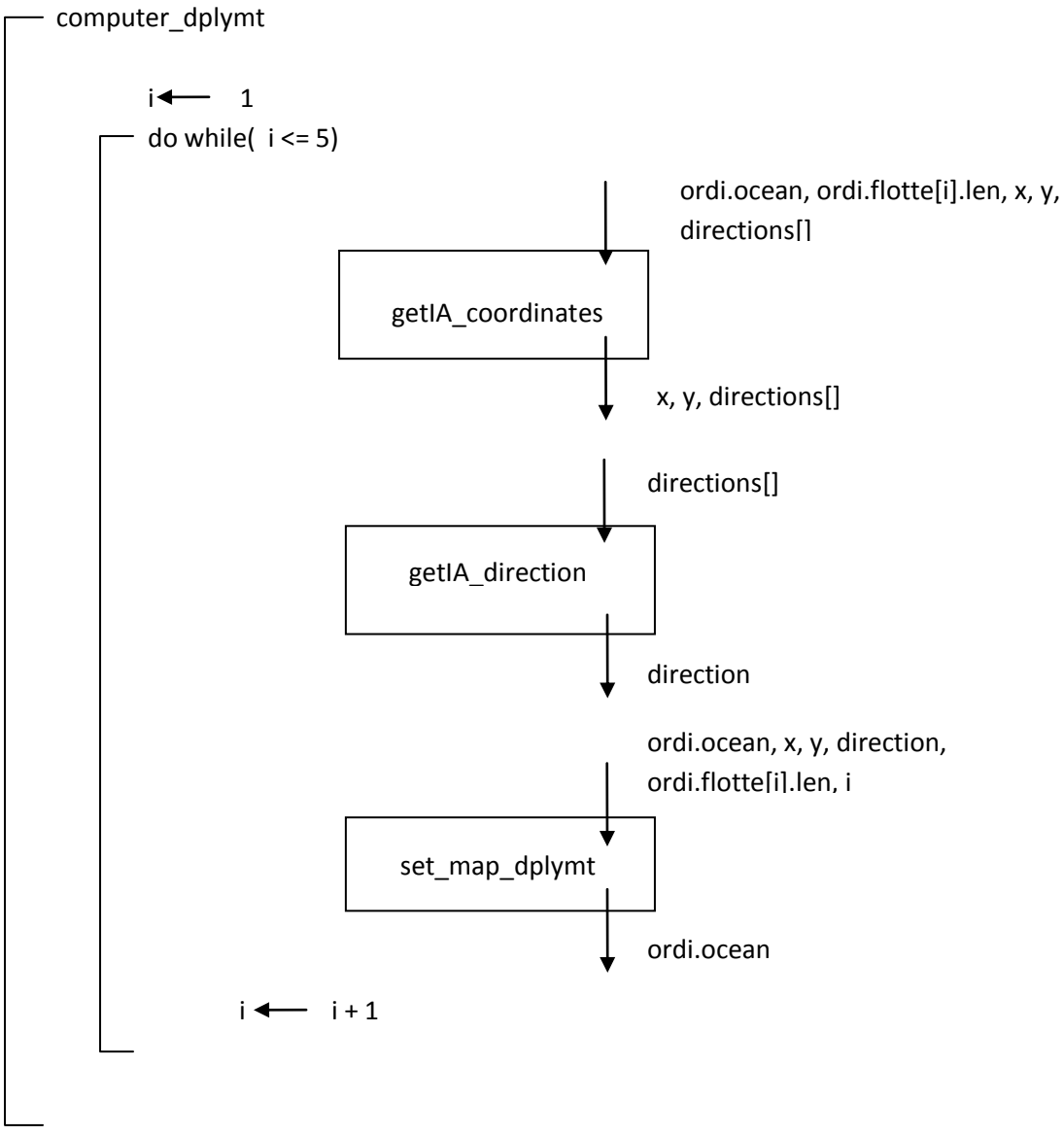
Sous-programme de : deploy_boat (Profondeur de niveau 2, niveau actuel : 3).

Paramètre en entrée/sortie : computer

Entête : computer_dplymt(var ordi : profil)

Pré-condition : ordi est complètement initialisé.

Post-condition : le champ « ocean » d'ordi est modifié selon la post-condition de set_map_dplymt.



GETIA_COORDINATES

Sous-programme de : computer_dplymt (Profondeur de niveau 3, niveau actuel : 4).

Paramètres en entrée : ordi.ocean, ordi.flotte[i].len

Paramètres en entrée/sortie : x, y, directions[]

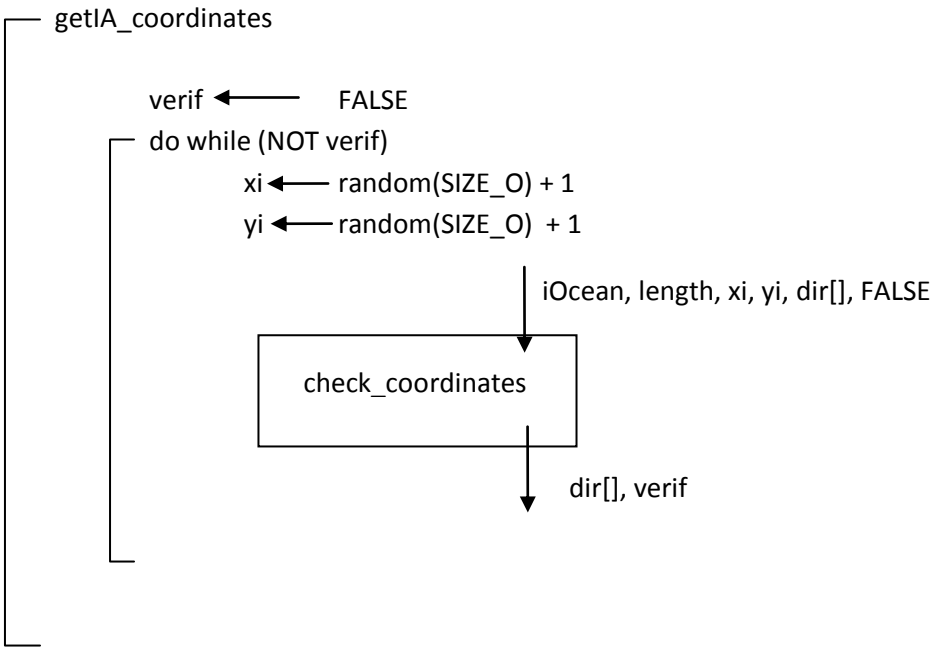
Entête : getIA_coordinates(var iOcean : tocean, length : integer, var xi : integer, var yi : integer, var dir : array[1..4] of boolean)

Pré-conditions :

- iOcean et length (contient la taille du bateau) sont complètement initialisés.
- xi, yi et dir sont déclarés.

Post-conditions :

- iOcean est inchangés.
- xi et yi sont initialisés avec des coordonnées valides selon les critères de vérification de check_coordinates.
- dir est initialisé selon la post-condition de check_coordinates.



GETIA_DIRECTION

Sous-programme de : computer_dplymt (Profondeur de niveau 3, niveau actuel : 4).

Paramètre en entrée : directions[]

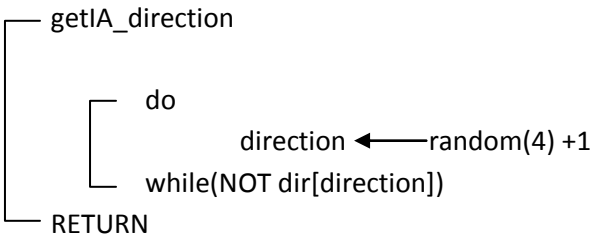
Paramètre en sortie : direction

Entête : getIA_direction(var dir : array[1..4] of boolean) direction : integer

Pré-condition : dir est complètement initialisé selon la post-condition de check_coordinates.

Post-condition : dir est inchangé.

Résultat : direction contient 1 si le Nord est la direction choisie, 2 si c'est l'est, 3 si c'est le sud et 4 si c'est l'ouest.



PLAYER_ROUND

Sous-programme de : Naval_Battle (Profondeur de niveau 1, niveau actuel : 2).

Paramètre en entrée/sortie : computer

Entête : player_round(var computr : profil)

Pré-condition : computr est initialisée et les bateaux sont déployés selon la post-condition de deploy_boat.

Post-condition :

- computr est modifié, le champ destroy du champ ocean est passé à false selon le choix de l'utilisateur et s'il n'y était pas déjà. Si il a touché un bateau on décrémente le champ lenRemain de flotte[bateauTouché] de 1. Si il coule le bateau on décrémente le champ boatRemain de 1.

Table des variables :

a) verif : booléen permettant de vérifier que les coordonnées sont bien dans l'océan.

Commentaire : L'utilisateur bénéficie de moins d'aide que l'ordinateur, on pourrait imaginer implémenter des niveaux de difficultés et aider l'utilisateur de la même façon que l'ordinateur, comme une piste à la réflexion.

player_round

ecrire("Océan de l'adversaire ! ")

computr.ocean

print_opponent_ocean

verif ← false

do while(NOT verif)

ecrire("Navigateur : Veuillez entrer la coordonnée de la zone a torpillée <A1 a ", 'A'+SIZE_O,SIZE_O, "> : ")

x, y

read_coordinates

x, y

verif ← x > 0 AND x <= SIZE_O AND y > 0 AND y <= SIZE_O

IF (NOT computr.ocean[y,x].destroy)

computr, x, y

set_attack

computr

ELSE

ecrire("Cannonier : Vous aviez déjà torpillé cette case ! Faites plus attention à l'avenir ! ")

PRINT_OPPONENT_OCEAN

Sous-programme de : player_round (Profondeur de niveau 2, niveau actuel : 3).

Paramètre en entrée : computr.ocean

Entête : print_opponent_ocean(var oceanToPrint : tocean)

Pré-condition : oceanToPrint est initialisé et les bateaux sont déployés selon la post-condition de deploy_boat.

Post-condition : oceanToPrint est inchangé, l'océan de l'adversaire est affiché, un '.' représente une case non torpillée, un '°' une case torpillée mais vide (à l'eau) et '*' un bateau touché.

Commentaire : Le joueur ne connait pas le nom des bateaux touchés grâce à cette visualisation, ce n'est pas très important pour le déroulement du jeu mais le nom du bateau est tout de même communiqué dans set_attack.

```
print_ocean

    i ⟵ 1
    do while(i <= SIZE_O)
            ecrire (i)
            i ⟵ i + 1
    i ⟵ 1
    do while( i<= SIZE_O)
            ecrire(64+i)
            j ⟵ 1
        do while( j <= SIZE_O)
            IF (NOT oceanToPrint[i,j].destroy)
                    ecrire('.')
            ELSE
                IF(oceanToPrint[i,j].etat <= 0)
                        ecrire('°')
                ELSE
                        ecrire('*')
                j ⟵ j + 1
    i⟵ i + 1

    ecrire("Légende : ")
    ecrire("Torpillé mais à l'eau : ° ")
    ecrire("Bateau touché : * ")
    ecrire("Cases non torpillées : . ")
```

SET_ATTACK

Sous-programme de : player_round (Profondeur de niveau 2, niveau actuel : 3).

Paramètres en entrée : x, y

Paramètre en entrée/sortie : computr

Entête : set_attack(var ordi : profil, xi : integer, yi : integer)

Pré-conditions :

- ordi est initialisé et les bateaux sont déployés selon la post-condition de deploy_boat.
- xi et yi contiennent les coordonnées valides de la case a torpillée.

Post-conditions :

- xi et yi sont inchangés.
- ordi est modifié :
    - Le champs ocean est modifié : la case en [yi,xi] est torpillée, le champ destroy de cette case est mis à TRUE.
    - Si cette case est un bateau, la longueur restante de celui-ci est mise à jour.
    - Si cette case coule un bateau, le nombre de bateau restant est décrémenté de 1.

```
set_attack

        ordi.ocean[yi,xi].destroy ⟵ true
        state ⟵ ordi.ocean[yi,xi].etat
    IF (state > 0)
            ordi.flotte[state].lenRemain ⟵ ordi.flotte[state].lenRemain – 1
        IF (ordi.flotte[state].lenRemain > 0)
                ecrire("Cannonier : Vous venez de touché le ",ordi.flotte[state].nom, "
                        de l'adversaire.")
        ELSE
                ecrire("Cannonier : Vous venez de coulé le ", ordi.flotte[state].nom,
                        "de l'adversaire. Exellente stratégie !")
                ordi.boatRemain ⟵ ordi.boatRemain - 1


    ELSE
            ecrire("Cannonier : Vous venez de torpillé une case vide ! Reprenez-vous
                commandant ! ")
```

COMPUTER_ROUND

Sous-programme de : Naval_Battle (Profondeur de niveau 1, niveau actuel : 2).

Paramètres en entrée/sortie : player, attack_module

Entête : computer_round(var gamer : profil, var attack_mod: IA_module)

Pré-conditions :

- gamer est initialisé et les bateaux sont déployés selon la post-condition de deploy_boat.
- attack_mod est initialisé.

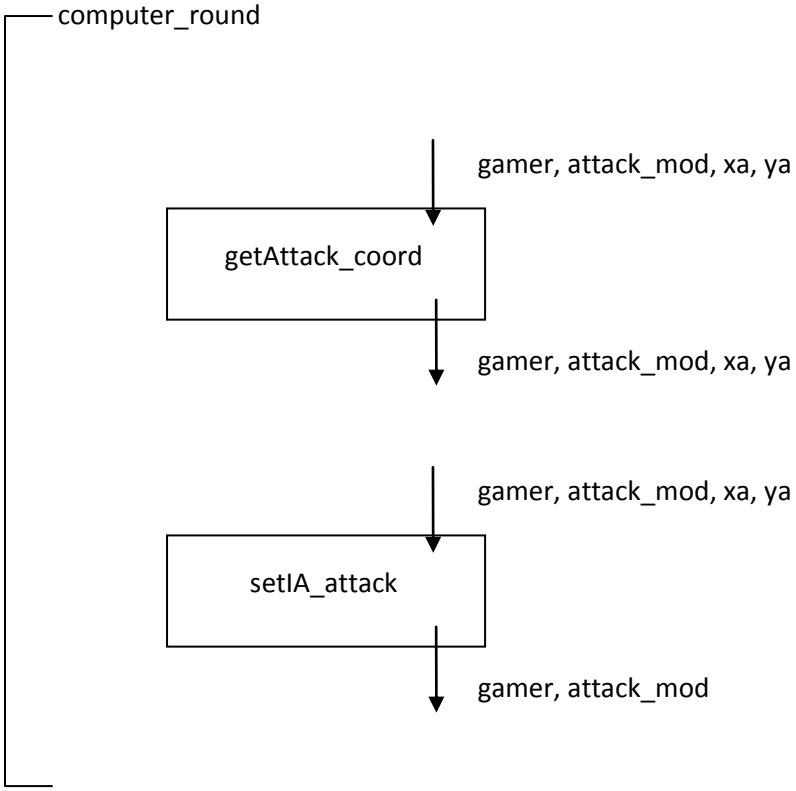
Post-conditions :

- gamer est modifié, le champ destroy du champ ocean[y,x] (où x et y sont les coordonnées choisies par l'ordinateur) est passé à false. Si il a touché un bateau on décrémente le champ lenRemain de flotte[bateauTouché] de 1. Si il coule le bateau on décrémente le champ boatRemain de 1.
- attack_mod est modifié : Si le champ boatFound est égal à TRUE :
    - Et si l'ordinateur a coulé un bateau, on passe boatFound à FALSE

  Sinon :

    - Et si l'ordinateur vient de touché un bateau on passe boatFound à TRUE, et on stock les coordonnées de cette case dans les champs xi et yi.

  Sinon attack_mod est inchangé.



GETATTACK_COORD

Sous-programme de : computer_round (Profondeur de niveau 2, niveau actuel : 3).

Paramètres en entrée/sortie : gamer, attack_mod, xa, ya

Entête : getAttack_coord (var player : profil, var brain: IA_module, var x : integer, var y : integer)

Pré-conditions :

- x et y sont déclarés.
- player est initialisé et les bateaux sont déployés selon la post-condition de deploy_boat.
- brain est initialisé.

Post-conditions :

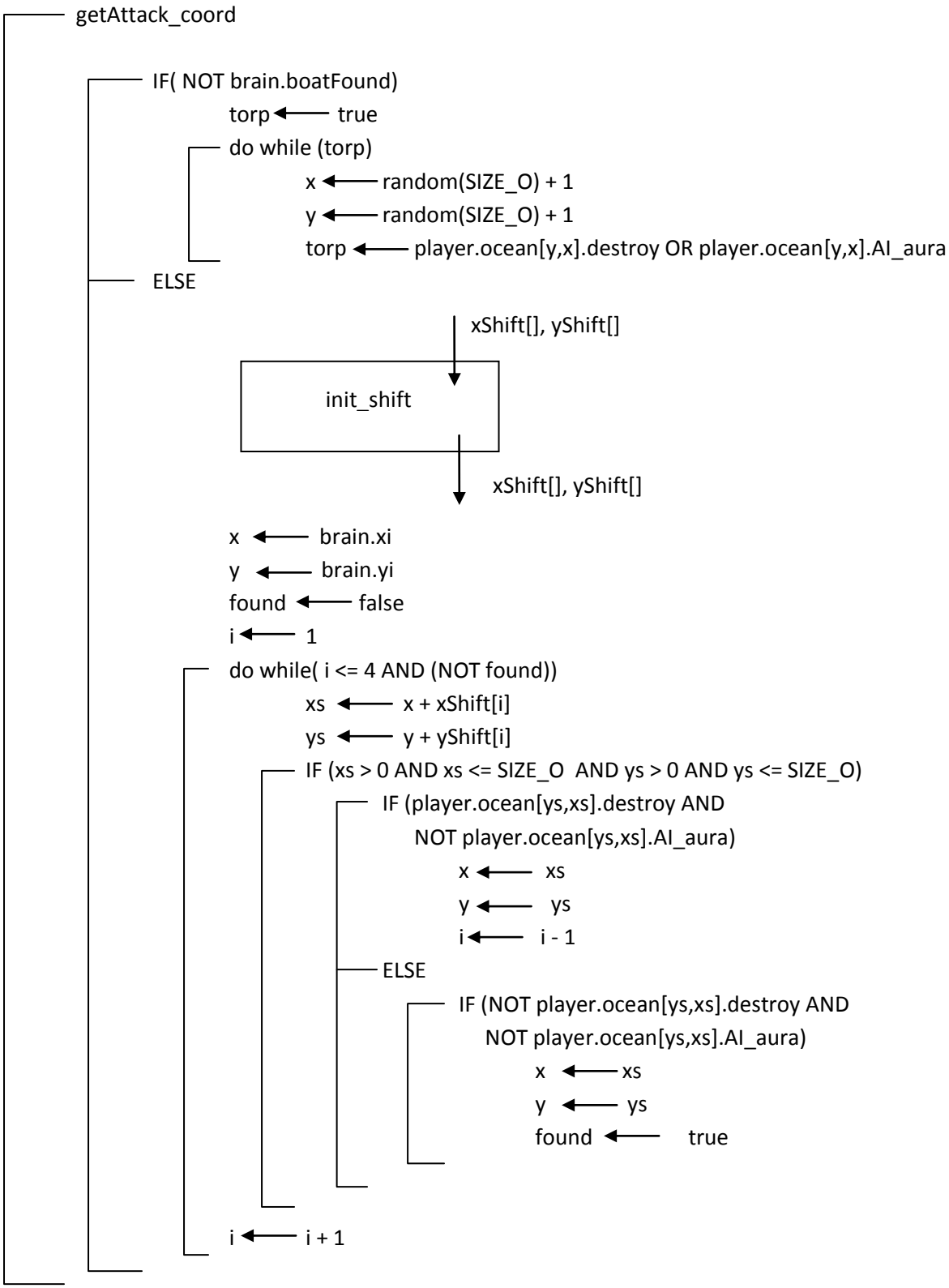
- player et brain sont inchangés.
- x et y sont initialisés aux coordonnées les plus judicieuses possibles, à savoir ne se trouvant pas sur une case déjà torpillée ou marquée comme 'aura'. Si un bateau est en cours de destruction, x et y seront initialisés à la prochaine case possible du bateau en question.

Table des variables :

a) torp : Permet de boucler tant que l'ordinateur n'a pas trouvé les coordonnées d'une case qui n'a pas déjà été touché et qui n'est pas une case d'aura.
b) xs et ys : Ce sont les coordonnées où peut potentiellement ce trouvé la prochaine case du bateau à détruire.

Commentaire :

Si un bateau ennemi n'est pas en cours de destruction, on recherche une case au hasard non torpillée et non marquée par AI_aura à TRUE. Si nous avons touché un bateau le coup précédent nous allons essayer de trouver les prochaines cases.
Si c'est la deuxième case que nous cherchons il nous faut donc trouver si le bateau est placé verticalement ou horizontalement. Dans notre algorithme et en général dans les suivant, lorsque nous voulons tourner autour d'une case : nous commençons toujours par le nord et puis tournons vers la droite (est-sud-ouest). De cette façon nous ne ratons aucunes cases.
Toutes les cases détruites seront marqués par deux signes : les champs de l'ocean : AI aura et destroy à TRUE, une exception est faite pour les bateaux touchés qui eux ne sont marqués que avec destroy à TRUE.
Dès lors, si nous tournons et que nous tombons sur une case avec une configuration de AI_aura à FALSE et destroy à TRUE, nous « switchons » dessus et recommençons l'opération. Nous décrémentons i de 1, et nous gardons la bonne direction. De cette façon, nous avons un moyen assez simple de tourner autours du bateau pour chercher les prochaines cases. Grâce aux cases marquées par AI_aura dans setIA_attack, nous n'attaquerons jamais une case dont nous pouvons logiquement admettre libre de bateau.
Evidemment la case trouvée n'a pas 100% de chance d'être la bonne. Avant de trouver la première case nous testerons au maximum 3 directions, dans ce cas les prochains coups seront tous gagnants vu que nous connaitrons la direction du bateau.


getAttack_coord
IF(NOT brain.boatFound)
torp ← true
do while (torp)
x ← random(SIZE_O) + 1
y ← random(SIZE_O) + 1
torp ← player.ocean[y,x].destroy OR player.ocean[y,x].AI_aura
ELSE
xShift[], yShift[]
init_shift
xShift[], yShift[]
x ← brain.xi
y ← brain.yi
found ← false
i ← 1
do while(i <= 4 AND (NOT found))
xs ← x + xShift[i]
ys ← y + yShift[i]
IF (xs > 0 AND xs <= SIZE_O AND ys > 0 AND ys <= SIZE_O)
IF (player.ocean[ys,xs].destroy AND
NOT player.ocean[ys,xs].AI_aura)
x ← xs
y ← ys
i ← i - 1
ELSE
IF (NOT player.ocean[ys,xs].destroy AND
NOT player.ocean[ys,xs].AI_aura)
x ← xs
y ← ys
found ← true
i ← i + 1

SETIA_ATTACK

Sous-programme de : computer_round (Profondeur de niveau 2, niveau actuel : 3).

Paramètres en entrée : xa, ya

Paramètres en entrée/sortie : gamer, attack_mod

Entête : setIA_attack (var player : profil, var brain: IA_module, x : integer, y : integer)

Pré-conditions :

- x et y sont initialisés.
- player est initialisé et les bateaux sont déployés selon la post-condition de deploy_boat.
- brain est initialisé.

Post-conditions :

- x et y sont inchangés.
- player et brain sont modifiés :
    - le champ destroy de player.ocean[y,x] est mis à true.
    - Si aucun bateau n'est présent sur cette case, alors le champ IA_aura de player.ocean[y,x] est également mis à true[5].
    - Sinon la longueur du bateau présent est décrémentée de 1.
        - Les champs xi et yi sont initialisés à x et y.
        - Si c'est la première case d'un bateau que l'ordinateur découvre on met le champ boatFound de brain à true.
        - Si ce n'est pas la première case, on place une aura autour de cette case (voir post-condition de set_aura).
            - Si le bateau est coulé, on décrémente boatRemain de 1 et appelons set_aura_border pour mettre les champs AI_aura des coordonnées de chaque extrémité du bateau à true. On ré-initialise brain.boatFound à false.

Table des variables :

a) state : C'est le champ « etat » de la coordonnée [y,x], pour une compréhension plus aisée de l'algorithme.
b) lenDestroy : C'est la longueur que l'ordinateur a déjà détruite du bateau.

[5] L'utilité d'une mise à false conditionnel de AI_aura est expliquée dans le commentaire de getAttack_coord.

```
setIA_attack
    player.ocean[y,x].destroy ⟵ true
    ecrire("L'ordinateur vient de vous torpiller aux coordonnées : ",64+y,x, ".")
    state ⟵ player.ocean[y,x].etat
    IF (state < 1)
        ecrire("Centre de contrôle : L'adversaire vient de torpillé une case vide, il
            semble faible mon commandant !")
        player.ocean[y,x].AI_aura ⟵ true

    ELSE
        player.flotte[state].lenRemain ⟵ player.flotte[state].lenRemain – 1
        lenDestroy ⟵ player.flotte[state].len - player.flotte[state].lenRemain
        IF (lenDestroy = 1)
            brain.boatFound ⟵ true
            ecrire("Centre de contrôle : Votre ", player.flotte[state].nom, "vient d'être touché ! ")
        ELSE
                                  player.ocean, x, y, brain.xi
                                  ↓
                              ┌──────────┐
                              │ set_aura │
                              └──────────┘
                                  ↓
                                  player.ocean

            IF(player.flotte[state].lenRemain > 0)
                ecrire("Centre de contrôle : Votre ", player.flotte[state].nom, "vient d'être touché ! ")
            ELSE
                ecrire("Centre de contrôle : Votre ", player.flotte[state].nom,"vient d'être coulé! ")
                player.boatRemain ⟵ player.boatRemain - 1
                                  player.ocean, x, y,
                                  player.flotte[state].len
                                  ↓
                              ┌──────────────┐
                              │ set_aura_border │
                              └──────────────┘
                                  ↓
                                  player.ocean

                brain.boatFound ⟵ false

        brain.xi ⟵ x
        brain.yi ⟵ y
```

SET_AURA_BORDER

Sous-programme de : setIA_attack (Profondeur de niveau 3, niveau actuel : 4).

Paramètres en entrée : x, y, player.flotte[state].len

Paramètre en entrée/sortie : player.ocean

Entête : set_aura_final (var oceanToSet : tocean, xa : integer, ya : integer, length : integer)

Pré-conditions :

- xa, ya et length sont initialisés.
- oceanToSet est initialisé et les bateaux sont déployés selon la post-condition de deploy_boat.

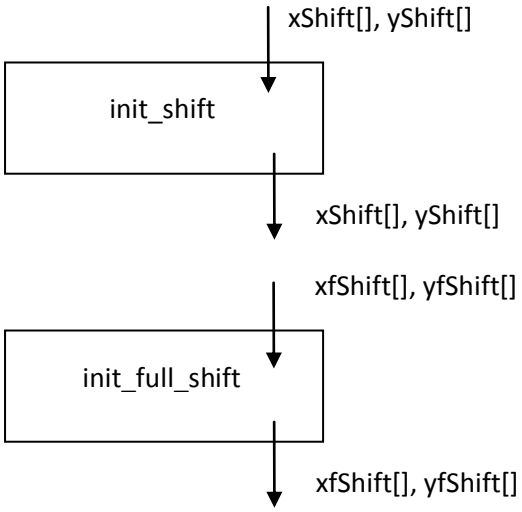
Post-conditions :

- oceanToSet est modifié : les champs AI_aura des deux extrémités du bateau sont mises à true car le bateau à été coulé, par conséquent on peut admettre que c'est l'aura du bateau.
- xa, ya et length sont inchangés.

Commentaire :

Dans un premier temps, nous cherchons la coordonnée opposée à xa et ya, c'est l'autre extrémité du bateau. Ensuite dans une même boucle nous tournons autour de ces deux cases et marquons toutes les cases libres avec AI_aura à true. Ces cases représentent l'aura et ne seront plus à torpillées.

set_aura_border




```
found ⟵ false
j ⟵ 1
do while (j<=4 AND NOT found)
        xs ⟵ xa + xShift[j]
        ys ⟵ ya + yShift[j]
        IF (xs > 0 AND xs <= SIZE_O  AND ys > 0 AND ys <= SIZE_O)
                IF (oceanToSet[ys,xs].destroy AND NOT oceanToSet[ys,xs].AI_aura)
                        xb ⟵ xa + (length – 1) * xShift[j]
                        yb ⟵ ya + (length – 1) * yShift[j]
                        found ⟵ true
        j ⟵ j + 1
j ⟵ 1
do while (j<=8)
        xs ⟵ xa + xfShift[j]
        ys ⟵ ya + yfShift[j]
        IF (xs > 0 AND xs <= SIZE_O  AND ys > 0 AND ys <= SIZE_O)
                IF (NOT oceanToSet[ys,xs].destroy AND NOT oceanToSet[ys,xs].AI_aura)
                        oceanToSet[ys,xs].AI_aura ⟵ TRUE
        xs ⟵ xb + xfShift[j]
        ys ⟵ yb + yfShift[j]
        IF (xs > 0 AND xs <= SIZE_O  AND ys > 0 AND ys <= SIZE_O)
                IF (NOT oceanToSet[ys,xs].destroy AND NOT oceanToSet[ys,xs].AI_aura)
                        oceanToSet[ys,xs].AI_aura ⟵ TRUE
        j ⟵ j + 1
```

SET_AURA

Sous-programme de : setIA_attack (Profondeur de niveau 3, niveau actuel : 4).

Paramètres en entrée : x, y, brain.xi

Paramètres en entrée/sortie : player.ocean

Entête : set_aura (var iOcean : tocean, xa : integer, ya : integer, xV : integer)

Pré-conditions :

- xa, ya et xV sont initialisés.
- iOcean est initialisé et les bateaux sont déployés selon la post-condition de deploy_boat.

Post-conditions :

- iOcean est modifié, pour tout [y,x] –mis à part les deux [y,x] aux extrémités du bateau - jouxtant (même en diagonale) [y_0, x_0] le champ AI_aura est mis à true, signalant ainsi la non-présence d'un bateau.
- xa, ya et xV sont inchangés.

Table des variables :

a) xs et ys : Représente les coordonnées décalées sur lequel nous pouvons potentiellement placé l'aura(-1).
b) shiftX et shiftY : Représente le décalage à effectuer pour mettre le champ AI_aura de iOcean à true sans mettre les cases dans l'alignement du bateau à true : on bloquerait alors la recherche de futur cases (cf. commentaire getAttack_coord).

```
set_aura
    IF(xV = xa)
            shiftX[1] ⟵ 1
            shiftX[2] ⟵ 1
            shiftX[3] ⟵ 1
            shiftX[4] ⟵ -1
            shiftX[5] ⟵ -1
            shiftX[6] ⟵ -1

            shiftY[1] ⟵ -1
            shiftY[2] ⟵ 0
            shiftY[3] ⟵ 1
            shiftY[4] ⟵ 1
            shiftY[5] ⟵ 0
            shiftY[6] ⟵ -1
    ELSE

            shiftX[1] ⟵ 0
            shiftX[2] ⟵ 1
            shiftX[3] ⟵ 1
            shiftX[4] ⟵ 0
            shiftX[5] ⟵ -1
            shiftX[6] ⟵ -1

            shiftY[1] ⟵ -1
            shiftY[2] ⟵ -1
            shiftY[3] ⟵ 1
            shiftY[4] ⟵ 1
            shiftY[5] ⟵ 1
            shiftY[6] ⟵ -1

       i ⟵ 1
    do while(i <= 6)
            xs ⟵ xa + shiftX[i]
            ys ⟵ ya + shiftY[i]
        IF (xs <= SIZE_O AND xs >0 AND ys <= SIZE_O AND ys >0)
                iOcean[ys,xs].AI_aura ⟵ true
            i  ⟵ i + 1
```