Taquin

Dossier de conception et code

SAE 1.02 – Comparaison d'approches algorithmiques

Maxime MONTOURO – TD2 / TP4

Rémi GENTIL – TD2 / TP4

Année universitaire 2022-2023

Table des matières

Principe du jeu 1
Clarifications / addendum aux spécifications externes 2
Algorithme 2
Sous Programmes 2
Algorithme 3
Description des sous-problèmes 4
Dictionnaire des variables 4
Algorithme 5
Description des sous-problèmes 6
Dictionnaire des variables 6
Algorithme 7
Description des sous-problèmes 8
Dictionnaire des variables 8
Algorithme 9
Description des sous-problèmes 10
Dictionnaire des variables 10
Algorithme 11
Description des sous-problèmes 12
Dictionnaire des variables 12
Algorithme 13
Description des sous-problèmes 14
Dictionnaire des variables 14
Algorithme 15
Description des sous-problèmes 15
Dictionnaire des variables 16
Algorithme 17
Description des sous-problèmes 18
Dictionnaire des variables 18
Algorithme 19
Description des sous-problèmes 20
Dictionnaire des variables 20
Taquin 21
Algorithme 21
Description des sous-problèmes 21
Dictionnaire des variables 22
TITRE ALGO 23
Algorithme 23
Description des sous-problèmes 24
Dictionnaire des variables 24
TITRE ALGO 25
Description des sous-problèmes 27
Dictionnaire des variables 27
TITRE ALGO 28

Algorithme 28
Description des sous-problèmes 29
Dictionnaire des variables 29
TITRE ALGO 31
Algorithme 31
Description des sous-problèmes 32
Dictionnaire des variables 32
TITRE ALGO 33
Algorithme 33
Description des sous-problèmes 34
Dictionnaire des variables 34
SECTION JOUER 36
Algorithme 36
Description des sous-problèmes 37
Dictionnaire des variables 37
TITRE ALGO 38
Algorithme 39
Description des sous-problèmes 40
Dictionnaire des variables 40
TITRE ALGO 41
Algorithme 41
Description des sous-problèmes 43
Dictionnaire des variables 43
TITRE ALGO 44
Algorithme 44
Description des sous-problèmes 46
Dictionnaire des variables 46
TITRE ALGO 47
Algorithme 47
Description des sous-problèmes 48
Dictionnaire des variables 48
Etat de finalisation : 51
Code source 52

Principe du jeu

Le jeu duTaquin est un jeu de logique dans lequel le joueur doit résoudre un carré composé d'une suite chiffres aléatoires qu'il doit remettre dans l'ordre numérique. La taille du carré peut varier selon l'envie du joueur. Par défaut, le carré proposé au joueur a une taille de 5 par 5.

Un mode debug est aussi proposé au joueur avant de lancer sa partie. Celui-ci consiste à lui indiquer les mouvements à effectuer afin de le réaliser.

Le joueur doit donc mettre les chiffres de la grille de Taquin dans l'ordre numérique avec la "case vide" dans un des quatre angles du carré. Le joueur peut donc exécuter autant de combinaisons souhaitées cependant il ne peut pas sortir du carré.

À tout moment le joueur peut abandonner la partie en utilisant la touche prévue à cet effet. Une fois le Taquin réalisé, le jeu se termine par un message félicitant le joueur pour sa performance, puis s'arrête.

Description des éléments du jeu :

Nous allons générer une grille de Taquin sous la forme suivante : un tableau d'entiers à deux dimensions définies avec deux constantes (préalablement définit). Nous avons décidé, pour la case vide, de créer un sous-programme qui affichera une case vide dont la valeur du tableau sera 0. Etant donné que nous traitons le cas d'un tableau d'entiers.

Afin d'être sûr que le Taquin soit réalisable, nous avons décidé de générer automatiquement une grille ordonnée et de la mélanger nous-même. De ce fait, nous pouvons donc stocker les différents échanges de cases dans une variable qui nous permettra donc d'avoir la solution pour le mode Debug.

Ce jeu s'exécute dans un terminal de commande. Le joueur a la possibilité d'activer le mode debug au début de la partie qui lui donnera le chemin à suivre pour résoudre la grille et gagner la partie. Ceci dit, si l'utilisateur ne suis pas le chemin fournis, le mode debug ne s'adaptera pas en fonction de ses mouvements mais il doit forcément proposer un chemin de base valide. Le joueur a autant de coup que possible pour réaliser le Taquin il n'y a pas de minimum ni de limites. Les valeurs de déplacements à inscrire seront :

La valeur de la case du taquin souhaité à être déplacé suivi (sans espace) d'une des lettres suivantes pour le mouvement du déplacement :

h pour déplacer la case vers le haut
b pour déplacer la case vers le bas
g pour déplacer la case vers la gauche

D pour déplacer la case vers la droite

Soit un exemple concret : 12g

Ps : le mouvement sera considéré comme valide si la case vide se situe à la position demandée. Ainsi, si l'utilisateur saisie 12g, le mouvement sera valide si la case vide se situe à sa droite.

1

Clarifications / addendum aux spécifications externes

Pour élaborer ce jeu, nous n'avons pas eu besoins d'inclure de nouvelles clarifications aux spécifications externes car elles étaient suffisamment complètes et claires dans le sujet et convenait à nos idées et ambitions pour ce projet. Nous avons donc décider de ne pas en rajouter.

Pour l'organisation du jeu, un module Taquin a été créé. Celui-ci est composé de de fichier (un au format .cpp et un au format .h). Ce module permet une meilleure visibilité du code. Nous retrouvons 3 rubriques :

- Affichage de la grille
- Génération de la grille
- Modificateurs

Ces différentes rubriques sont composées chacune de fonctions et/ou procédures dont le but est spécifié directement dans le fichier.

Ainsi que deux variables globales nommées NB_LIGNE et NB_COLONNE. Ces variables permettent notamment de pouvoir générer la grille de taquin au début du jeu mais aussi d'avoir constamment accès aux dimensions de la grille.

Des sous programmes comme genererTaquin (qui permet de générer une grille de taquin ordonnée) mais aussi afficherGrilleTaquin se trouve dans le modules.

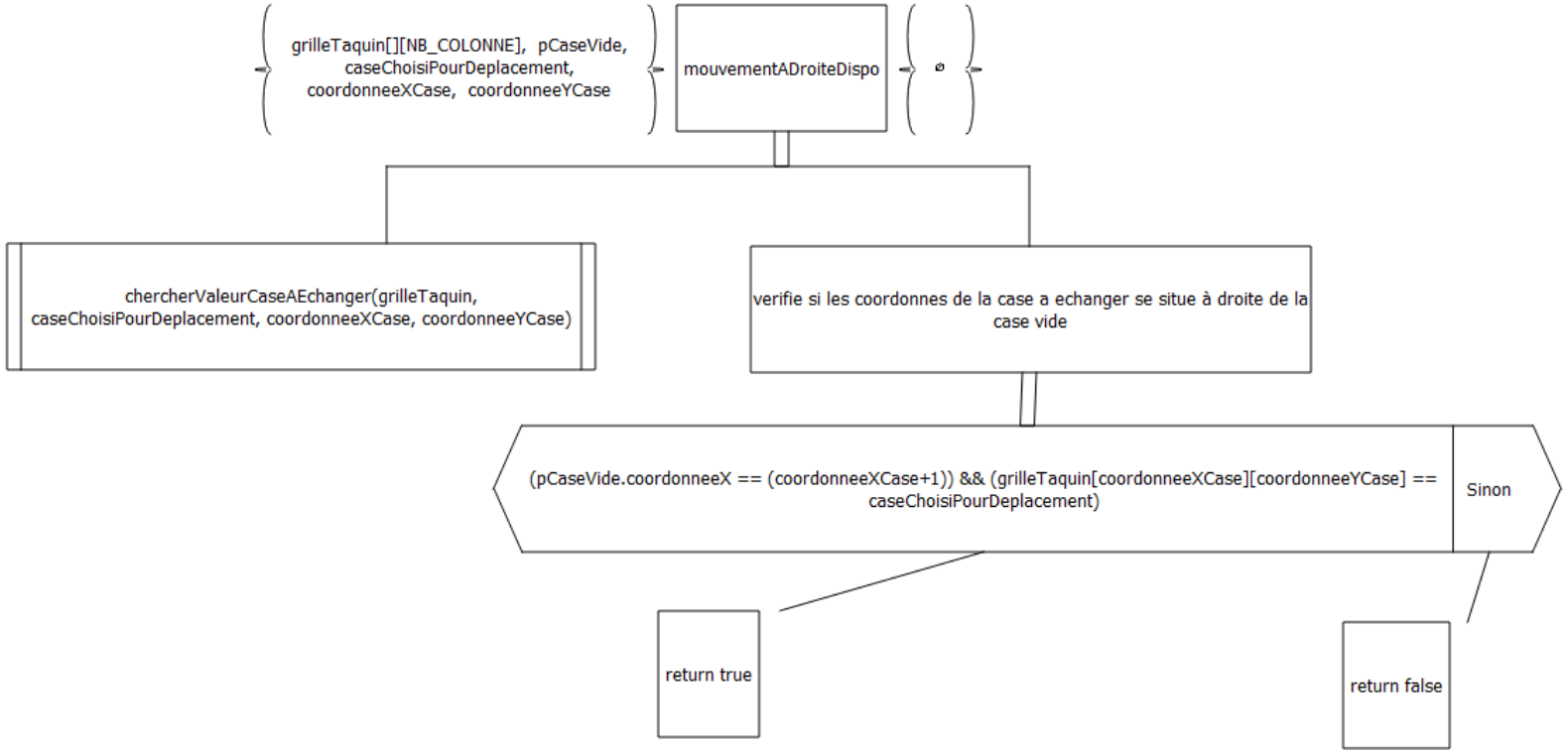
Algorithme

Sous Programmes

Dans cette rubrique nous retrouvons les différents sous programmes contenue dans le module taquin. Plus précisément ceux de la rubrique « MODIFICATEURS » . Ceux des autres rubriques sont définit dans les emplacements prévus à cette effet dans le document.

2

Algorithme



3

Description des sous-problèmes

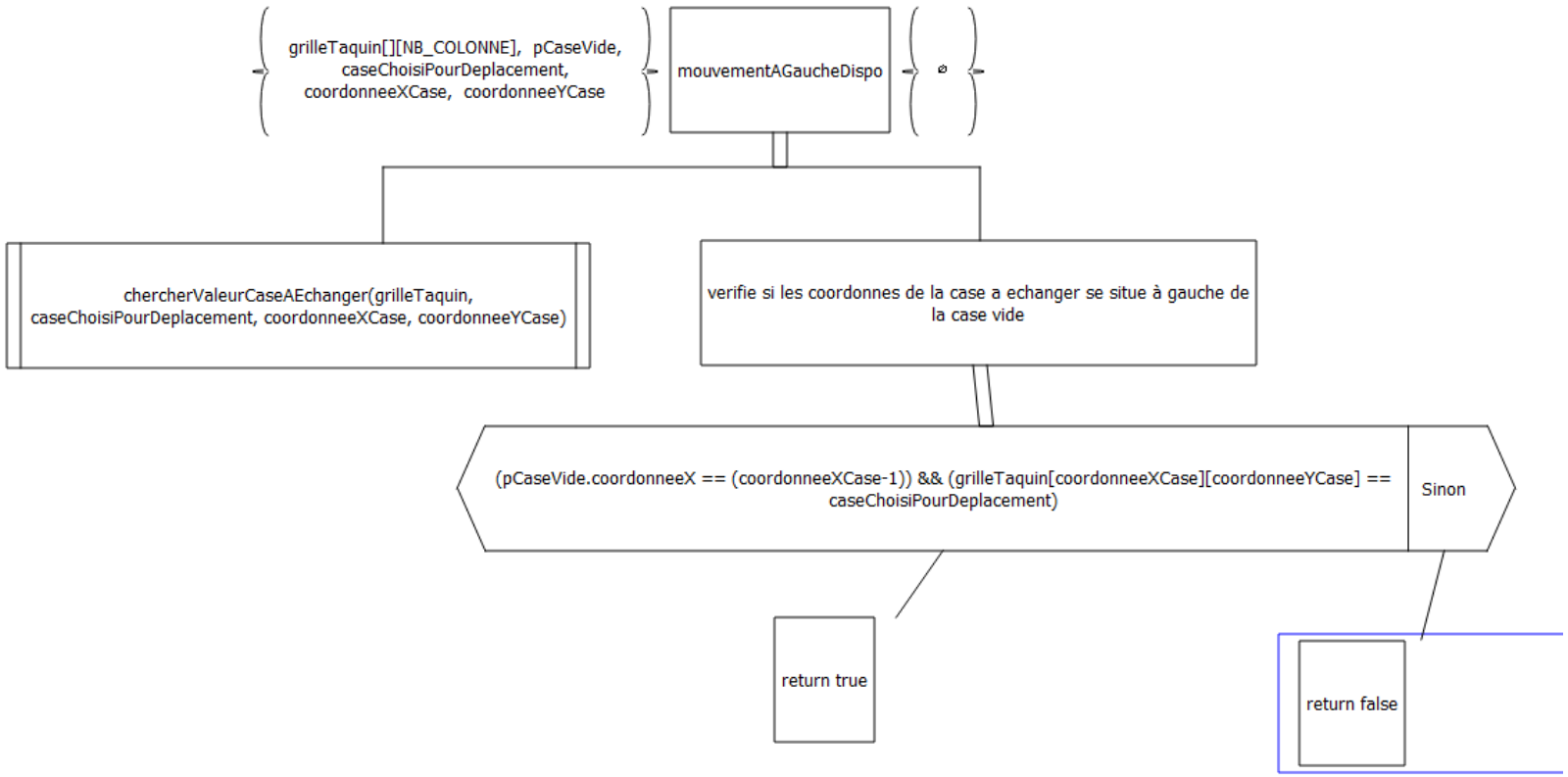
Cette algorithme cherche les indices de la case à échanger et vérifier si les coordonnées de cette case se situe bien à droite de la case vide. Si ce n'est pas le case alors le programme renverra faux

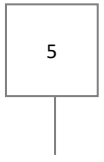
Dictionnaire des variables

Nom	**Type**	**Signification**
grilleTaquin[NB_LIGNE][NB_COLONNE]	int	Tableau d'entiers à deux dimensions de longueur NB_LIGNE et NB_COLONNE
pCaseVide	CaseVide	Coordonnées de la case vide
caseChoisiPourDeplacement	int	Case choisi pour le déplacement
coordonneeX	Bool	Coordonne X de la case à échanger
coordonneeY	Bool	Coordonne Y de la case à échanger

4

Algorithme


grilleTaquin[][NB_COLONNE], pCaseVide, caseChoisiPourDeplacement, coordonneeXCase, coordonneeYCase
mouvementAGaucheDispo
ø
chercherValeurCaseAEchanger(grilleTaquin, caseChoisiPourDeplacement, coordonneeXCase, coordonneeYCase)
verifie si les coordonnes de la case a echanger se situe à gauche de la case vide
(pCaseVide.coordonneeX == (coordonneeXCase-1)) && (grilleTaquin[coordonneeXCase][coordonneeYCase] == caseChoisiPourDeplacement)
Sinon
return true
return false


5

Description des sous-problèmes

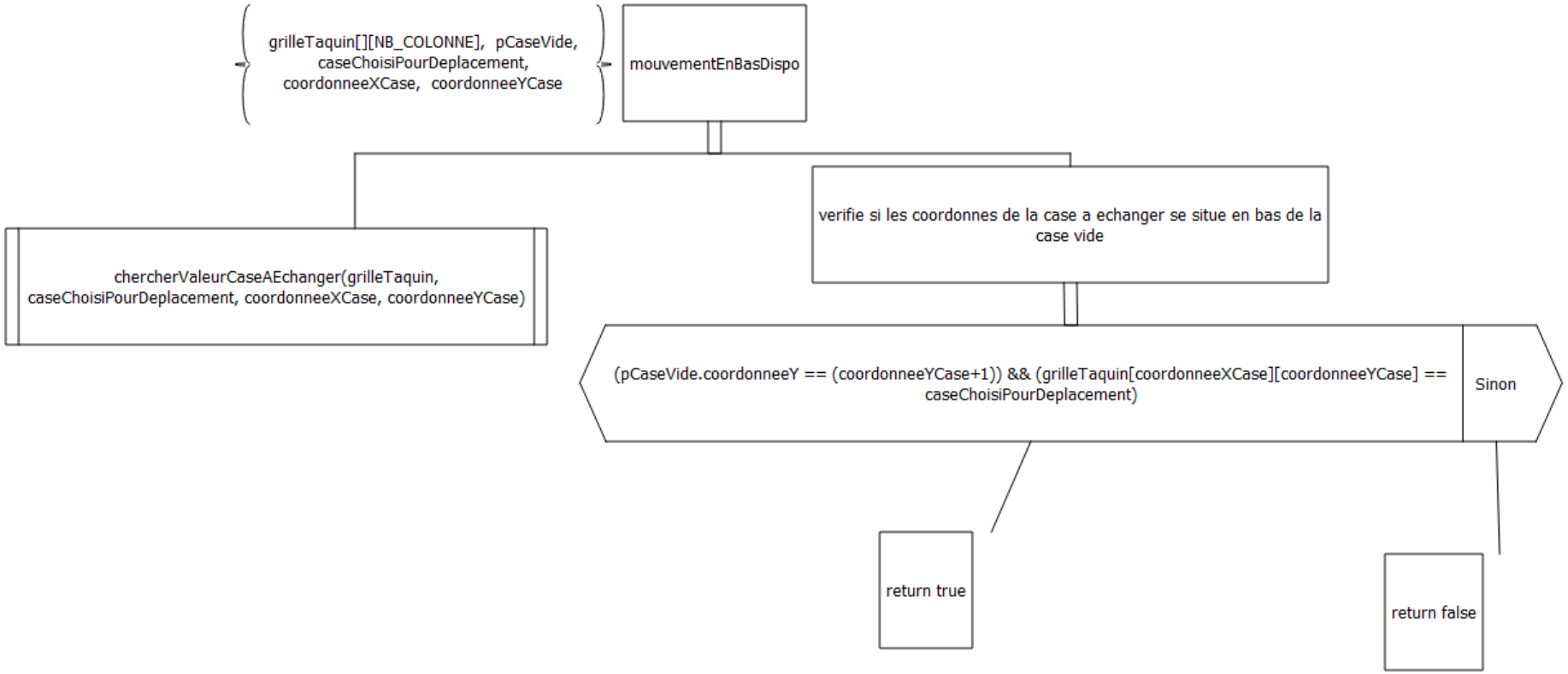
Cette algorithme cherche les indices de la case à échanger et vérifier si les coordonnées de cette case se situe bien à gauche de la case vide. Si ce n'est pas le case alors le programme renverra faux

Dictionnaire des variables

Nom	**Type**	**Signification**
grilleTaquin[NB_LIGNE][NB_COLONNE]	int	Tableau d'entiers à deux dimensions de longueur NB_LIGNE et NB_COLONNE
pCaseVide	CaseVide	Coordonnées de la case vide
caseChoisiPourDeplacement	int	Case choisi pour le déplacement
coordonneeX	Bool	Coordonne X de la case à échanger
coordonneeY	Bool	Coordonne Y de la case à échanger

6

Algorithme



7

Description des sous-problèmes

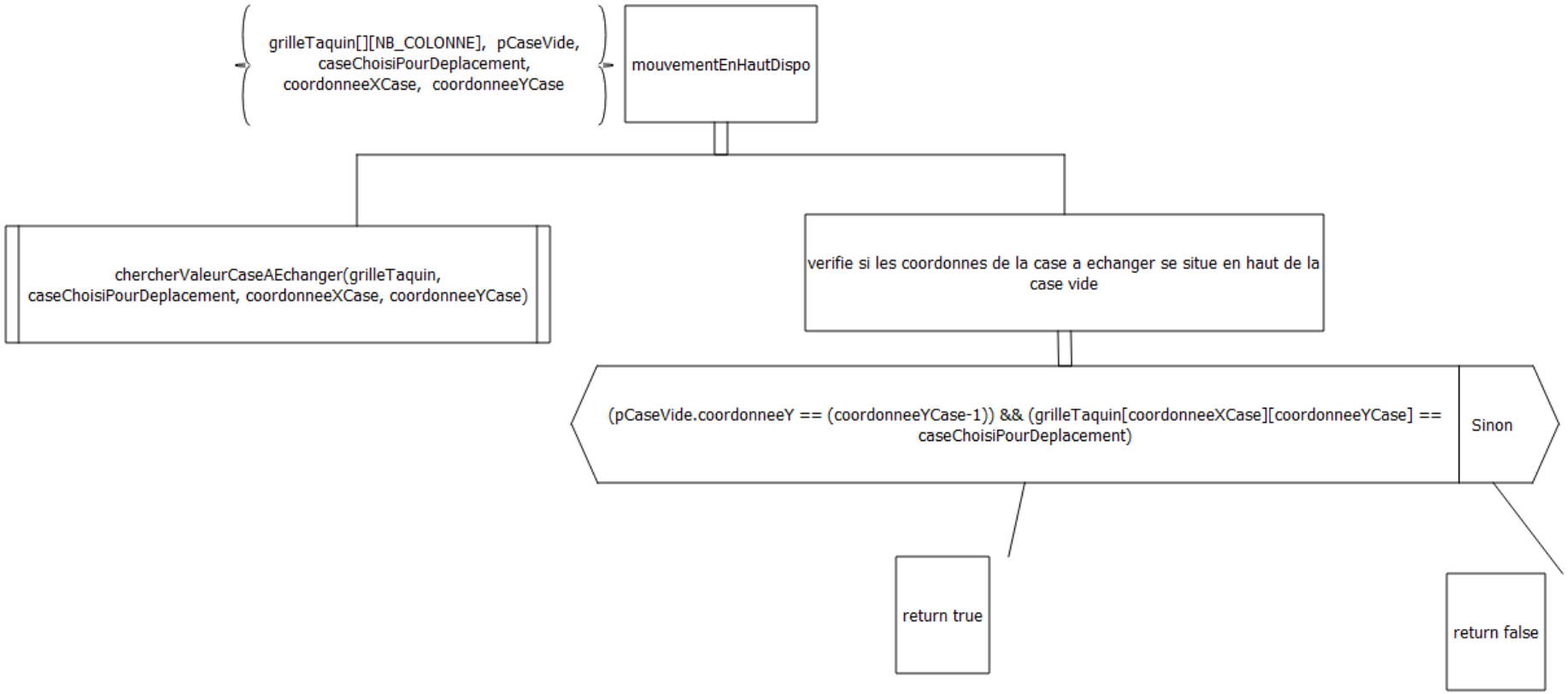
Cette algorithme cherche les indices de la case à échanger et vérifier si les coordonnées de cette case se situe bien en bas de la case vide. Si ce n'est pas le case alors le programme renverra faux

Dictionnaire des variables

Nom	**Type**	**Signification**
grilleTaquin[NB_LIGNE][NB_COLONNE]	int	Tableau d'entiers à deux dimensions de longueur NB_LIGNE et NB_COLONNE
pCaseVide	CaseVide	Coordonnées de la case vide
caseChoisiPourDeplacement	int	Case choisi pour le déplacement
coordonneeX	Bool	Coordonne X de la case à échanger
coordonneeY	Bool	Coordonne Y de la case à échanger

8

Algorithme



9

Description des sous-problèmes

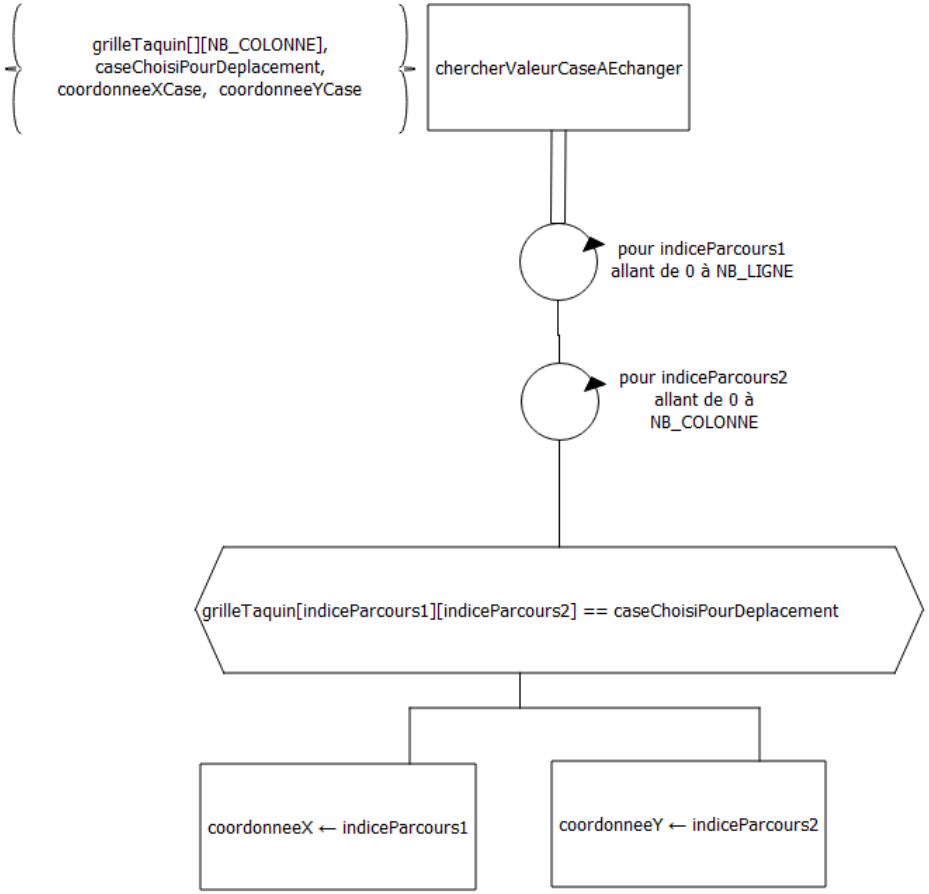
Cette algorithme cherche les indices de la case à échanger et vérifier si les coordonnées de cette case se situe bien en haut de la case vide. Si ce n'est pas le case alors le programme renverra faux

Dictionnaire des variables

Nom	**Type**	**Signification**
grilleTaquin[NB_LIGNE][NB_COLONNE]	int	Tableau d'entiers à deux dimensions de longueur NB_LIGNE et NB_COLONNE
pCaseVide	CaseVide	Coordonnées de la case vide
caseChoisiPourDeplacement	int	Case choisi pour le déplacement
coordonneeX	Bool	Coordonne X de la case à échanger
coordonneeY	Bool	Coordonne Y de la case à échanger

10

Algorithme


grilleTaquin[][NB_COLONNE],
caseChoisiPourDeplacement,
coordonneeXCase, coordonneeYCase
chercherValeurCaseAEchanger
pour indiceParcours1
allant de 0 à NB_LIGNE
pour indiceParcours2
allant de 0 à
NB_COLONNE
grilleTaquin[indiceParcours1][indiceParcours2] == caseChoisiPourDeplacement
coordonneeX ← indiceParcours1
coordonneeY ← indiceParcours2
11
11

Description des sous-problèmes

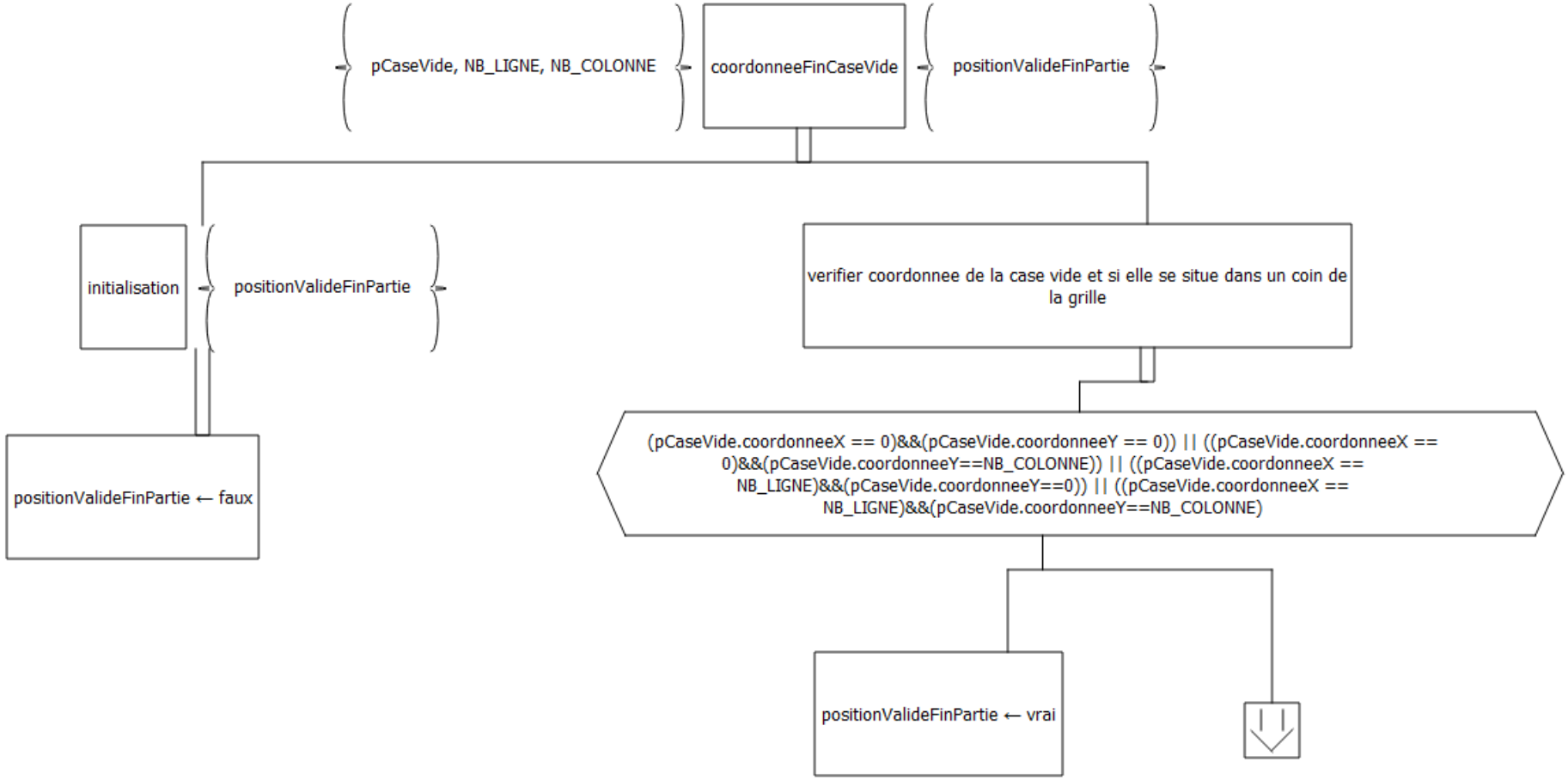
Cette algorithme cherche les indices de la case choisi par le joueur et affecte aux variables coordonneX et coordonneY les coordonnées de la valeur choisi par le joueur

Dictionnaire des variables

Nom	**Type**	**Signification**
grilleTaquin[NB_LIGNE][NB_COLONNE]	int	Tableau d'entiers à deux dimensions de longueur NB_LIGNE et NB_COLONNE
caseChoisiPourDeplacement	int	Case choisi pour le déplacement
coordonneeX	Bool	Coordonne X de la case à échanger
coordonneeY	Bool	Coordonne Y de la case à échanger

12

Algorithme



13

Description des sous-problèmes

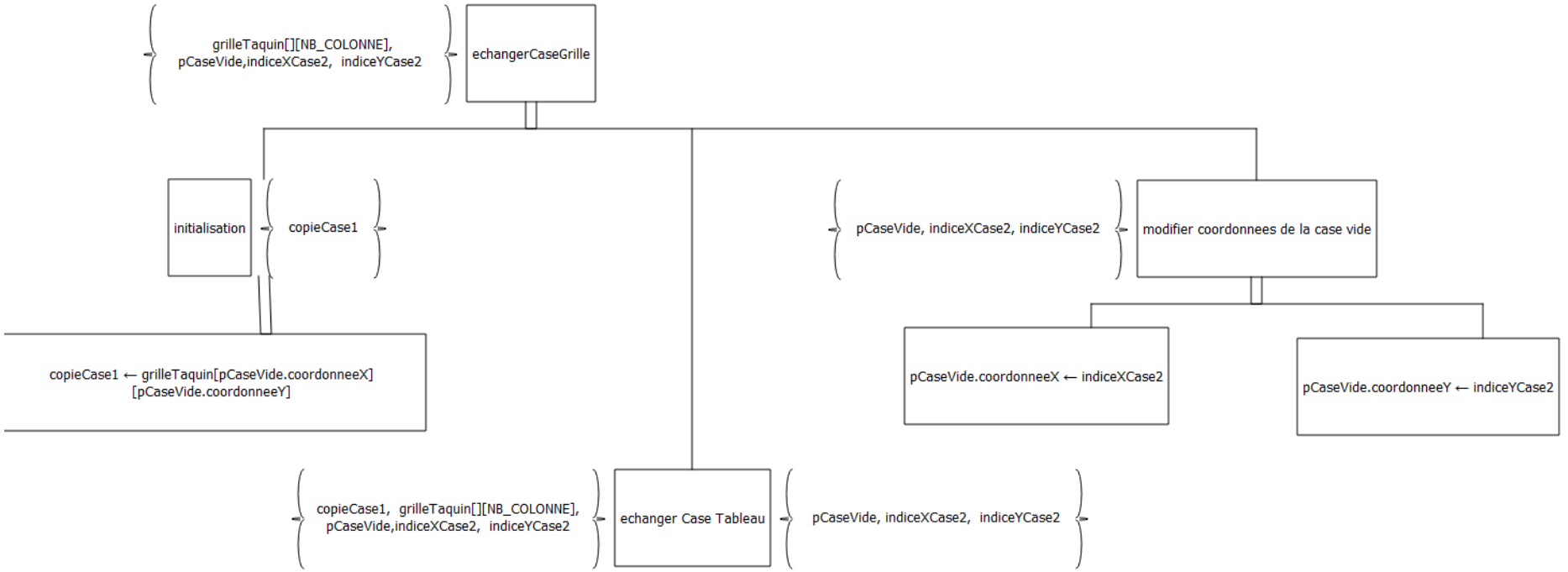
Cette algorithme vérifie si les coordonnées de la case vide à la fin de partie sont valides. En somme, si la case vide se situe dans un des quatre coins de la grille de taquin

Dictionnaire des variables

Nom	**Type**	**Signification**
grilleTaquin[NB_LIGNE][NB_COLONNE]	int	Tableau d'entiers à deux dimensions de longueur NB_LIGNE et NB_COLONNE
pCaseVide	CaseVide	Position de la case vide
NB_LIGNE	Const unsigned short int	Dimensions de la grille de taquin
NB_COLONNE	Const unsigned short int	Dipmension de la grille de taquin
positionValideFinPartie	bool	Renvoie vrai si la position de la case vide est valide, faux sinon

14

Algorithme



Description des sous-problèmes

Cette algorithme change les cases de la grille. Il est basé sur le modèle algorithme « échanger deux cases d'un tableau ». Par la suite les nouvelles coordonnées de la case vide sont associées aux valeurs de l'enregistrement.

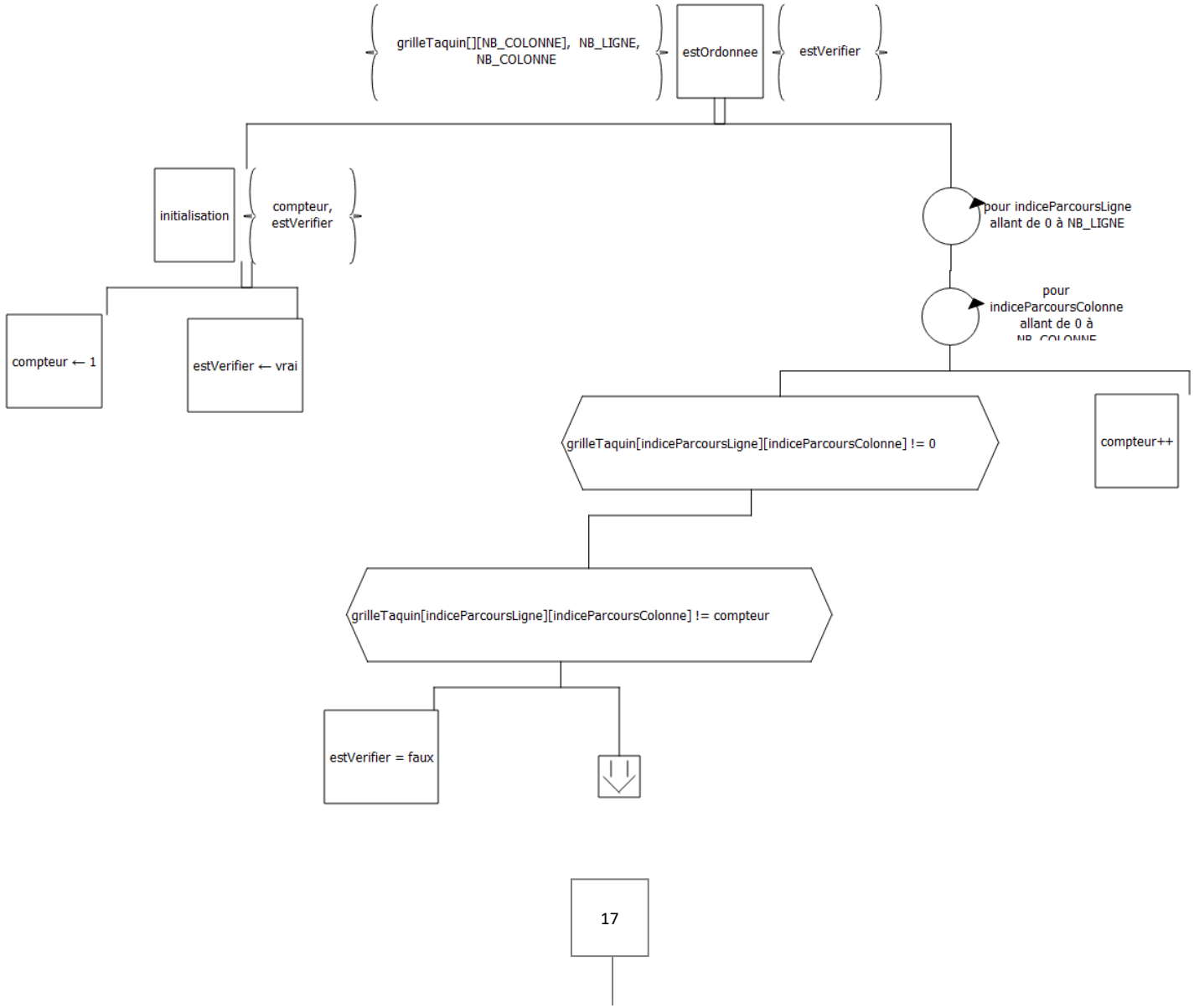
15

Dictionnaire des variables

Nom	Type	Signification
grilleTaquin[NB_LIGNE][NB_COLONNE]	int	Tableau d'entiers à deux dimensions de longueur NB_LIGNE et NB_COLONNE
copieCase1	int	Copie de la case 1 à echanger
coordonneeX	Bool	Coordonne X de la case à échanger
coordonneeY	Bool	Coordonne Y de la case à échanger

16

Algorithme


grilleTaquin[][NB_COLONNE], NB_LIGNE, NB_COLONNE
estOrdonnee
estVerifier
initialisation
compteur, estVerifier
compteur ← 1
estVerifier ← vrai
pour indiceParcoursLigne allant de 0 à NB_LIGNE
pour indiceParcoursColonne allant de 0 à
grilleTaquin[indiceParcoursLigne][indiceParcoursColonne] != 0
compteur++
grilleTaquin[indiceParcoursLigne][indiceParcoursColonne] != compteur
estVerifier = faux
17
17

Description des sous-problèmes

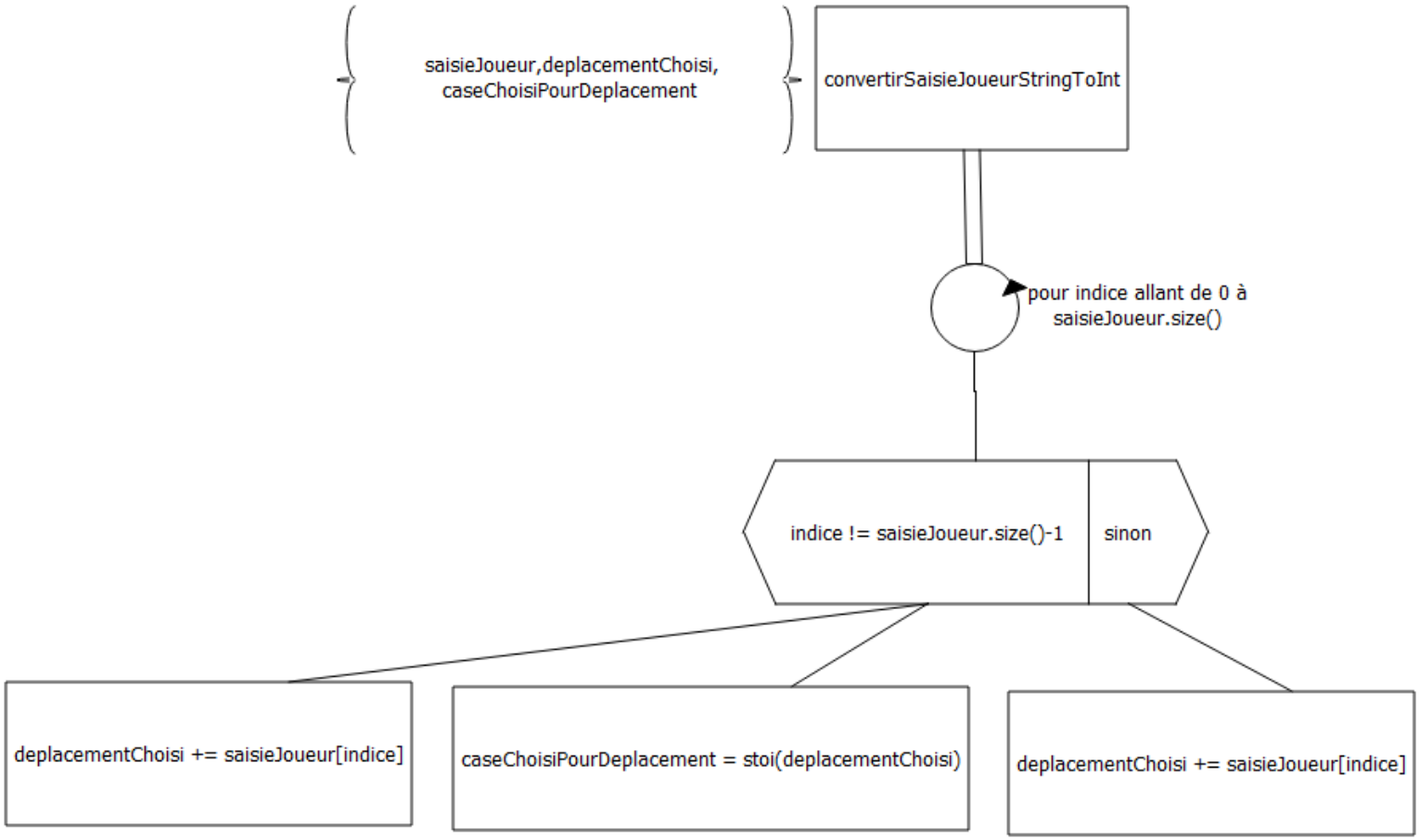
Cette algorithme vérifie si la grille de taquin est ordonnée à l'aide d'un compteur. Cette algorithme ne prend pas en compte la case vide (qui a pour valeur 0). Si la grille est ordonnée alors est renvoyé estVerifier qui est initialisé à vrai, faux sinon

Dictionnaire des variables

Nom	**Type**	**Signification**
grilleTaquin[NB_LIGNE][NB_COLONNE]	int	Tableau d'entiers à deux dimensions de longueur NB_LIGNE et NB_COLONNE
estVerifier	Bool	Renvoie vrai si la grille de taquin est ordonnée, faux sinon
indiceParcoursLigne	Unsigned short int	Indice de parcours des lignes du tableau
indiceParcoursColonne	Unsigned short int	Indice de parcours des colonnes du tableau

18

Algorithme



19

Description des sous-problèmes

Cette algorithme convertit la saisie du joueur d'une chaîne de caractère en une chaîne de caractère et la valeur de la case

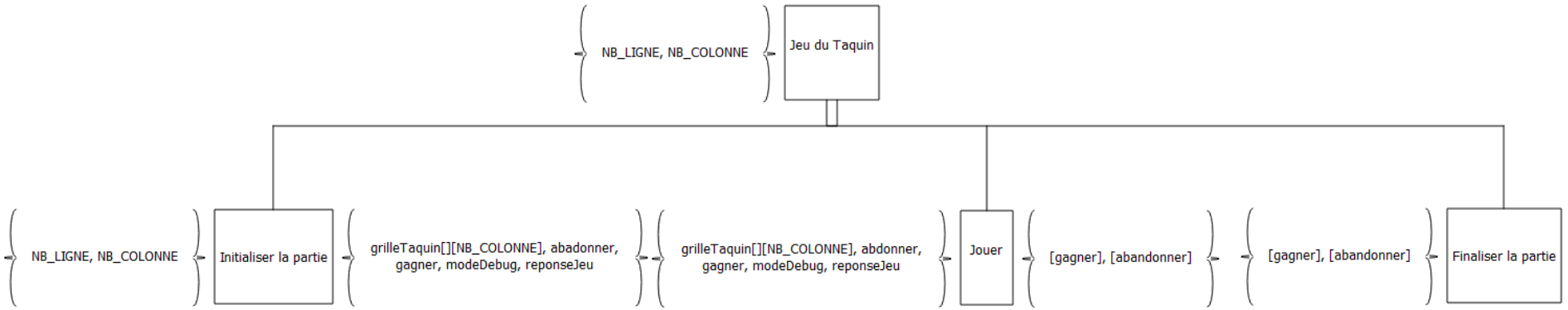
Dictionnaire des variables

Nom	**Type**	**Signification**
grilleTaquin[NB_LIGNE][NB_COLONNE]	int	Tableau d'entiers à deux dimensions de longueur NB_LIGNE et NB_COLONNE
deplacementChoisi	int	Déplacement choisi par le joueur
caseChoisiPOurDeplacement	Int	Valeur de la case choisi pour le déplacement
saisieJoueur	string	La saisie du joueur pendant la phase de choix

20

Taquin

Algorithme



Description des sous-problèmes

L'algorithme principal du jeu se décompose en 3 sous-problèmes principaux :

Initialiser la partie : Qui prend en compte les dimensions de la grille de Taquin et en résultat la grille de Taquin, les booléens gagner et abandonner, le booléen pour savoir si on affiche oui ou non le mode debug et la réponse à ce mode debug.

Jouer la partie : représente le fonctionnement du jeu avec en argument la grille de Taquin, les booléens gagner et abandonner, le booléen pour savoir si on affiche oui ou non le mode debug et la réponse à ce mode debug. En résultat, les booléens gagner et abandonner.

21

Finaliser la partie : représente la finalisation de la partie. Cette partie de l'algorithme prend en paramètre les booléens gagner et abandonner. Selon, la réponse des booléens, un message sera affiché à l'écran.

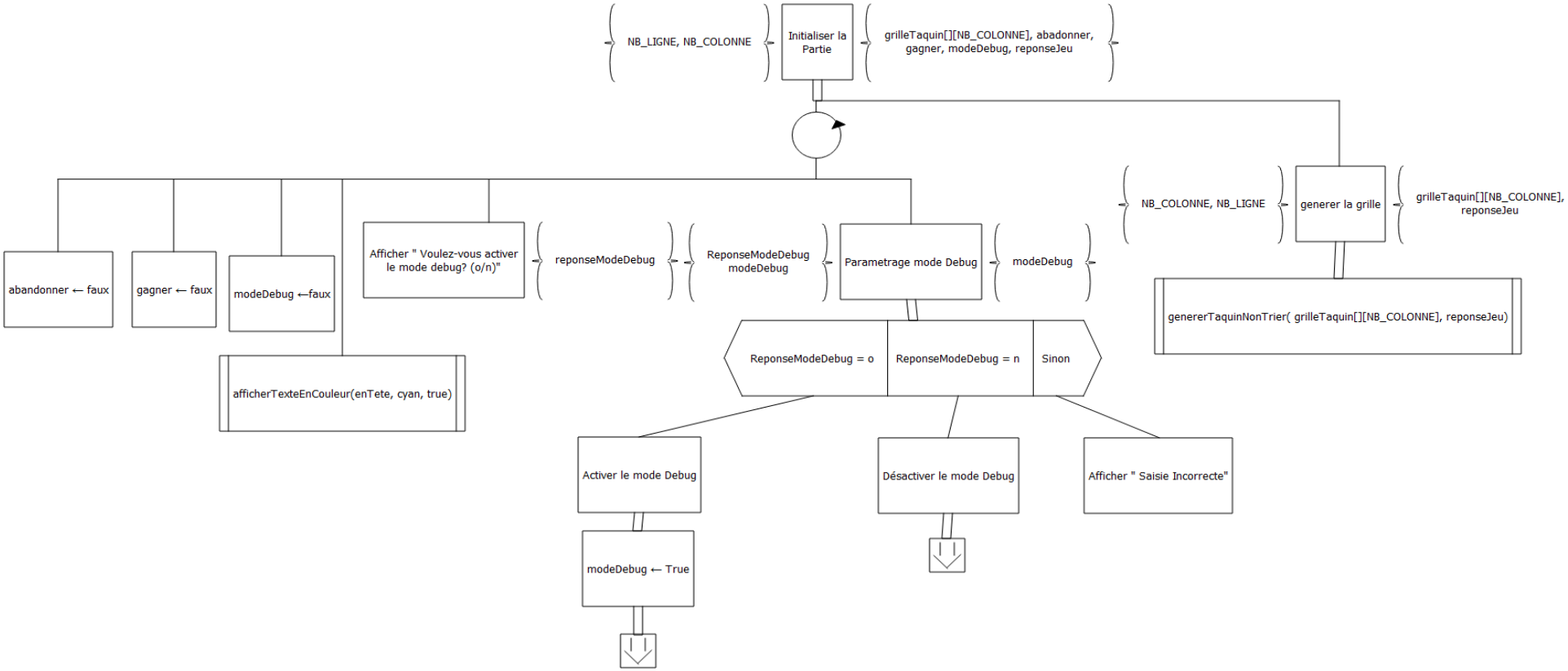
Dictionnaire des variables

Nom	**Type**	**Signification**
grilleTaquin[NB_LIGNE][NB_COLONNE]	int	Tableau d'entiers à deux dimensions de longueur NB_LIGNE et NB_COLONNE
NB_LIGNE	Const unsigned short int	Nombre de ligne du tableau
NB_COLONNE	Const unsigned short int	Nombre de colonne du tableau
gagner	Bool	Si le joueur a gagné la partie, initialisé à faux
abandonner	Bool	Si le joueur a abandonné la partie, initialisé à faux
modeDebug	Bool	Si le joueur a décidé d'activer le mode debug, initialisé à faux
reponseJeu	string	La solution pour résoudre la grille avec les mouvements à réaliser

22

TITRE ALGO

Algorithme



23

Description des sous-problèmes

Ce sous problème porte sur l'initialisation de la partie. Cette partie du problème porte sur l'initialisation de la grille avec le sous-programme genererTaquinNonTrier, mais aussi sur l'initialisation du mode debug.

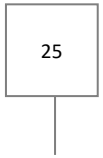
Dictionnaire des variables

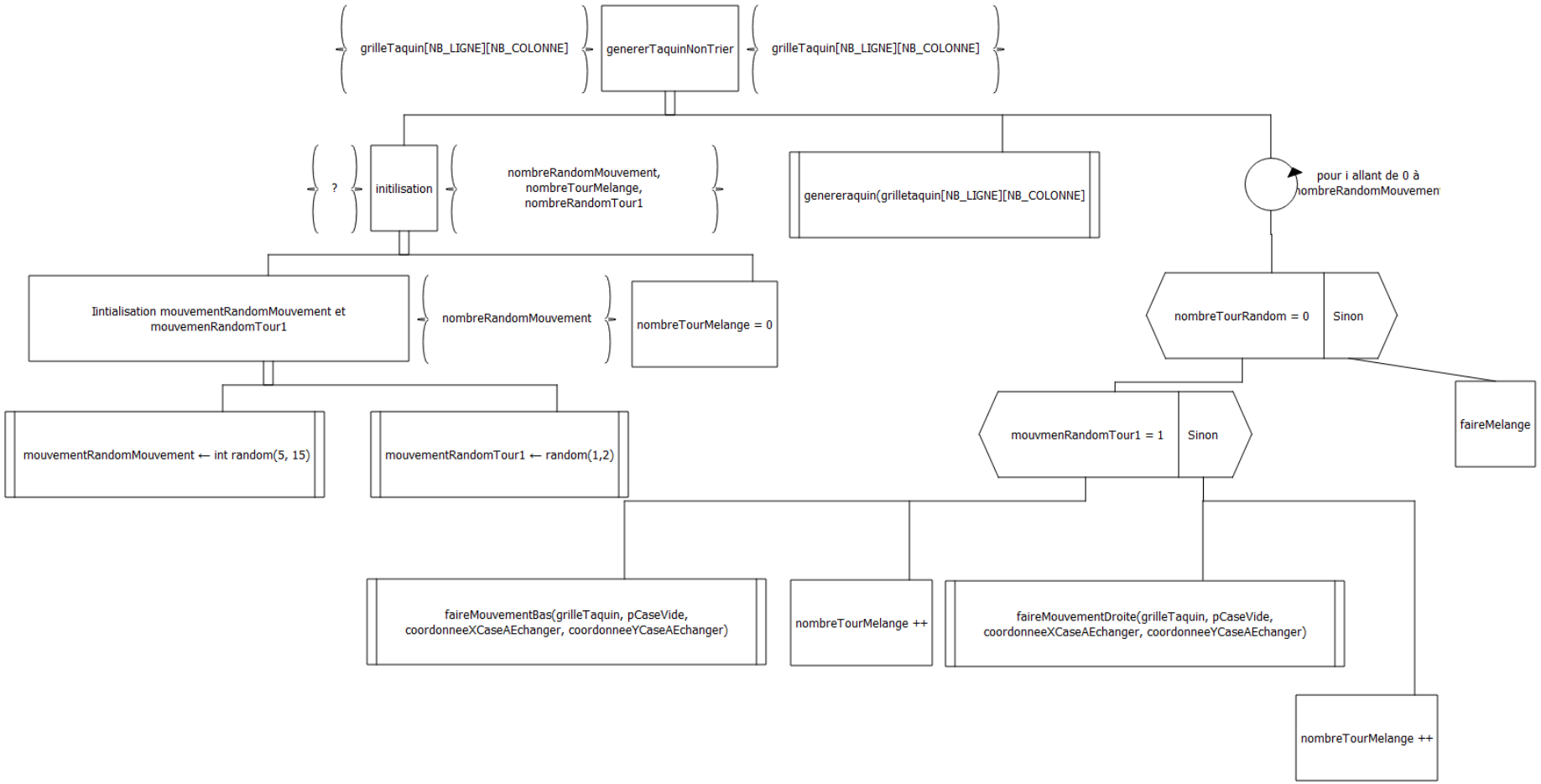
Nom	Type	Signification
grilleTaquin[NB_LIGNE][NB_COLONNE]	int	Tableau d'entiers à deux dimensions de longueur NB_LIGNE et NB_COLONNE
NB_LIGNE	Const unsigned short int	Nombre de ligne du tableau
NB_COLONNE	Const unsigned short int	Nombre de colonne du tableau
gagner	Bool	Si le joueur a gagné la partie, initialisé à faux
abandonner	Bool	Si le joueur a abandonné la partie, initialisé à faux
modeDebug	Bool	Si le joueur a décidé d'activer le mode debug, initialisé à faux
reponseJeu	string	La solution pour résoudre la grille avec les mouvements à réaliser

24

TITRE ALGO

Algorithme




grilleTaquin[NB_LIGNE][NB_COLONNE]
genererTaquinNonTrier
grilleTaquin[NB_LIGNE][NB_COLONNE]
?
initilisation
nombreRandomMouvement,
nombreTourMelange,
nombreRandomTour1
genereraquin(grilletaquin[NB_LIGNE][NB_COLONNE]
pour i allant de 0 à
Iintialisation mouvementRandomMouvement et
mouvemenRandomTour1
nombreRandomMouvement
nombreTourMelange = 0
nombreTourRandom = 0
Sinon
mouvementRandomMouvement ← int random(5, 15)
mouvementRandomTour1 ← random(1,2)
mouvmenRandomTour1 = 1
Sinon
faireMelange
faireMouvementBas(grilleTaquin, pCaseVide,
coordonneeXCaseAEchanger, coordonneeYCaseAEchanger)
nombreTourMelange ++
faireMouvementDroite(grilleTaquin, pCaseVide,
coordonneeXCaseAEchanger, coordonneeYCaseAEchanger)
nombreTourMelange ++

26

Description des sous-problèmes

Ce sous-programme génère une grille de taquin non triée. Afin d'obtenir la solution au jeu, une grille de taquin ordonnée est créé à l'aide du sous-programme genererTaquin. Cette algorithme est basé sur un certain nombre de sous programmes utilisé ici.

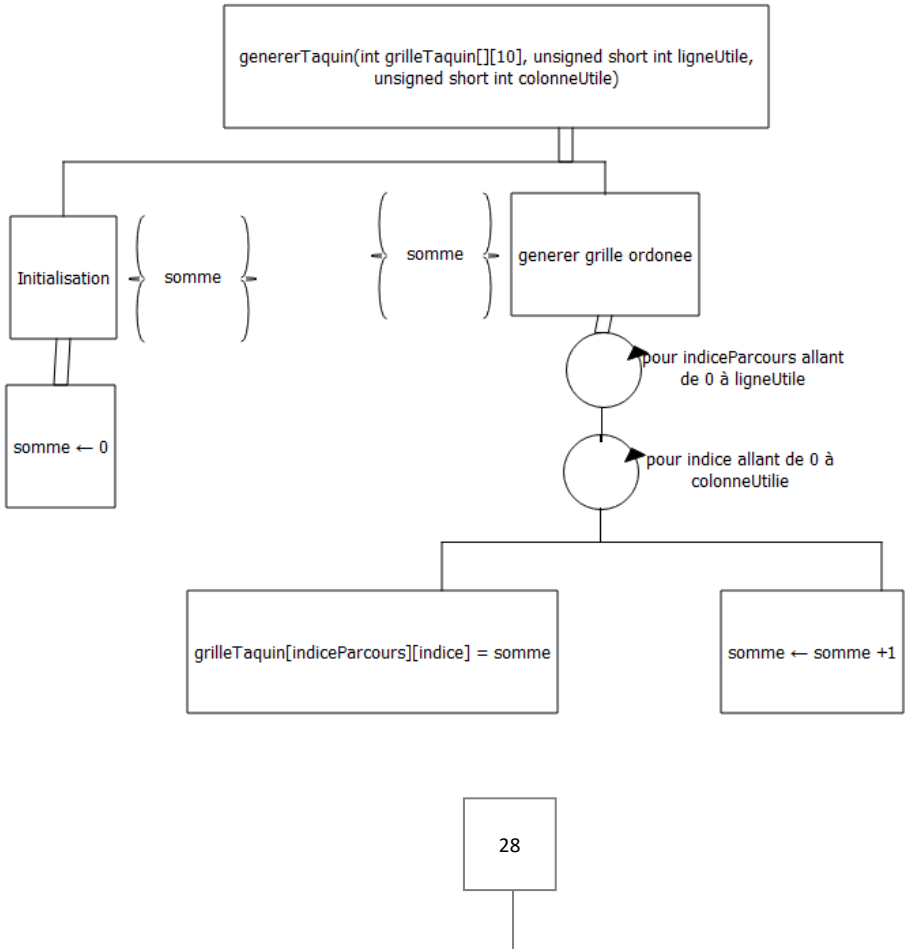
Dictionnaire des variables

Nom	Type	Signification
grilleTaquin[NB_LIGNE][NB_COLONNE]	int	Tableau d'entiers à deux dimensions de longueur NB_LIGNE et NB_COLONNE
NB_LIGNE	Const unsigned short int	Nombre de ligne du tableau
NB_COLONNE	Const unsigned short int	Nombre de colonne du tableau
mouvementRadom	int	Nombre aléatoire sur le nombre de mélange
nombreTourMelange	Int	Compteur d'itération du mélange
mouvementRandomTour1	Int	Le mouvement aléatoire fait au tour 1

27

TITRE ALGO

Algorithme



Description des sous-problèmes

Cet algorithme génère une grille de Taquin dans un tableau d'entiers à deux dimensions de dimensions NB_LIGNE et NB_COLONNE (définit préalablement dans le code) ces constantes peuvent être modifié afin de changer les dimensions de la grille selon les préférences du joueur. Un compteur ajoute la valeur N+1 à chaque itération dans la boucle.

Dictionnaire des variables

Nom	Type	Signification

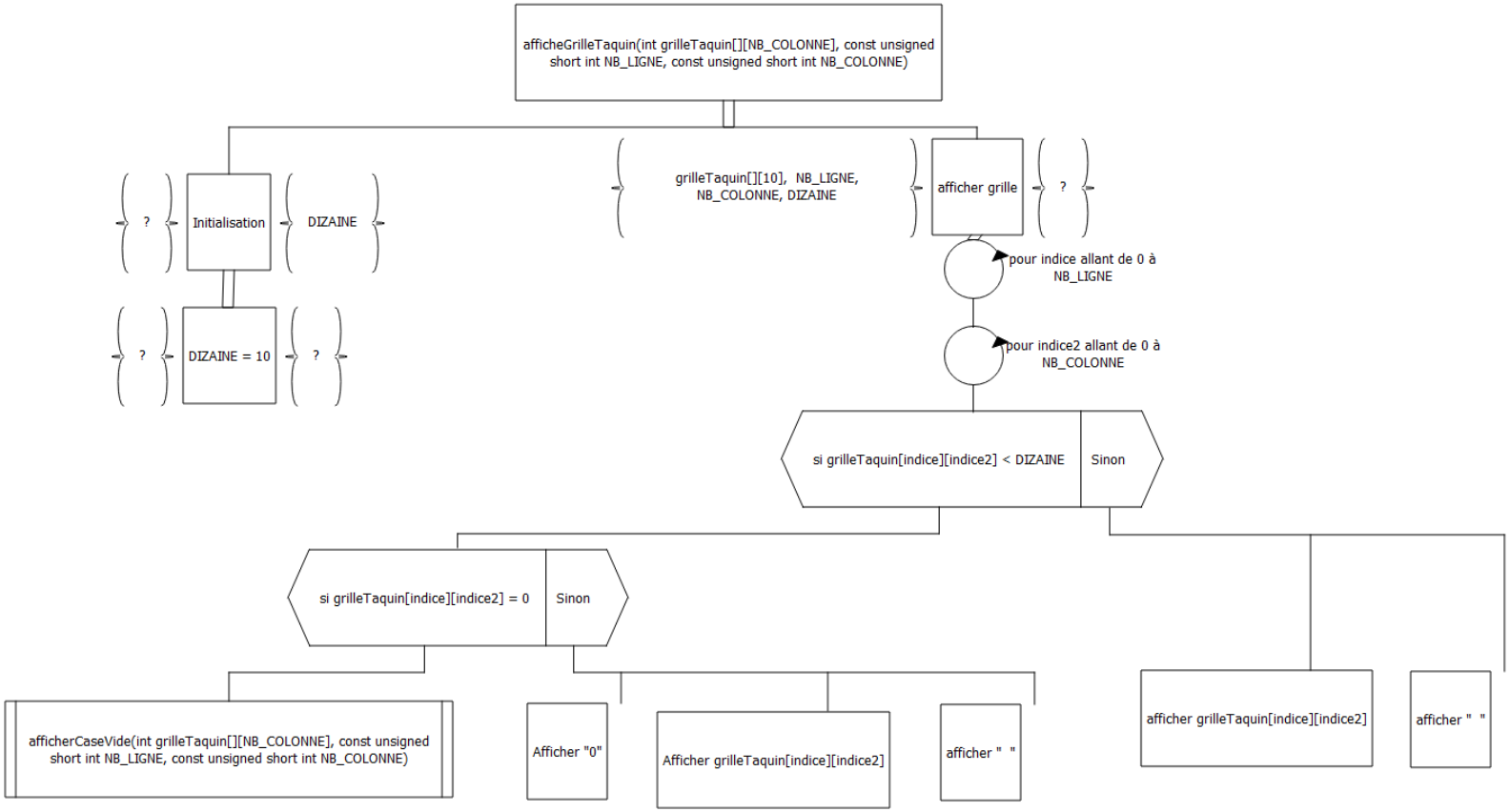
29

grilleTaquin[NB_LIGNE][NB_COLONNE]	int	Tableau d'entiers à deux dimensions de longueur NB_LIGNE et NB_COLONNE
NB_LIGNE	Const unsigned short int	Nombre de ligne du tableau
NB_COLONNE	Const unsigned short int	Nombre de colonne du tableau
somme	Int	Compteur qui permet d'ajouter les différentes valeurs dans les cases du tableau

30

TITRE ALGO

Algorithme



31

Description des sous-problèmes

Cet algorithme affiche la grille de Taquin avec certaines conditions. Si la valeur de la case du tableau est inférieure à 10 alors l'algorithme analyse si la valeur est 0. Si oui, il affiche une case vide sinon il affiche un 0 puis la valeur de la case afin que toutes les lignes du tableau soit aligné. Si la valeur du tableau est supérieure à 10 alors il affiche la valeur de la case du tableau.

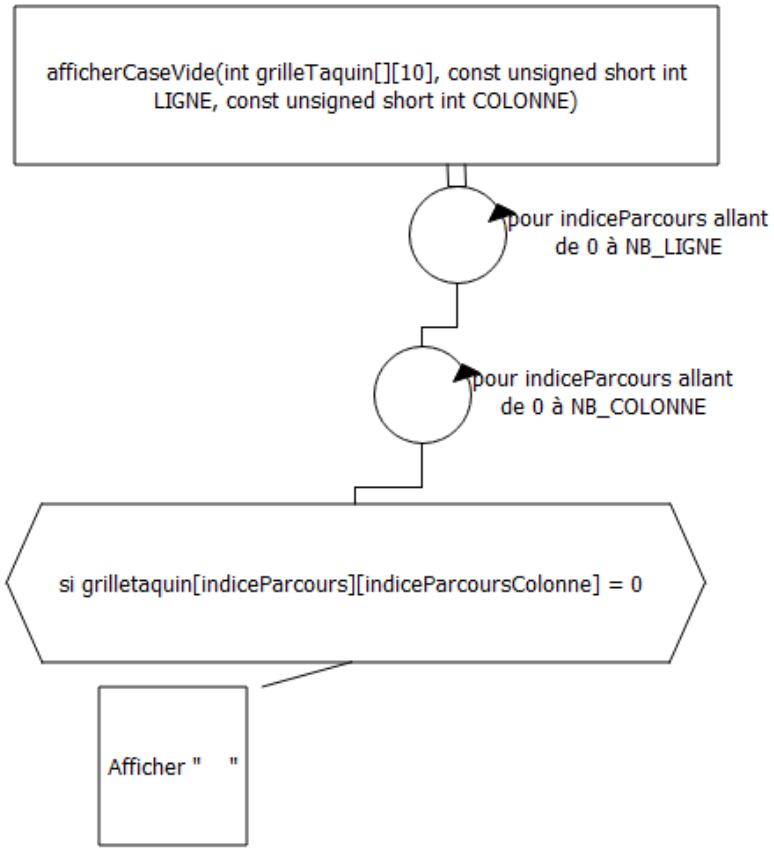
Dictionnaire des variables

Nom	Type	Signification
grilleTaquin[NB_LIGNE][NB_COLONNE]	int	Tableau d'entiers à deux dimensions de longueur NB_LIGNE et NB_COLONNE
NB_LIGNE	Const unsigned short int	Nombre de ligne du tableau
NB_COLONNE	Const unsigned short int	Nombre de colonne du tableau
DIZAINE	Const int	Variable correspondant au nombre 10 qui est untilisé pour des tests

32

TITRE ALGO

Algorithme



Description des sous-problèmes

Cet algorithme parcours la grille de taquin et affiche “ “ si la valeur de la case de la grille de taquin est 0.

Dictionnaire des variables

Nom	Type	Signification
grilleTaquin[NB_LIGNE][NB_COLONNE]	int	Tableau d’entiers à deux dimensions de longueur NB_LIGNE et NB_COLONNE

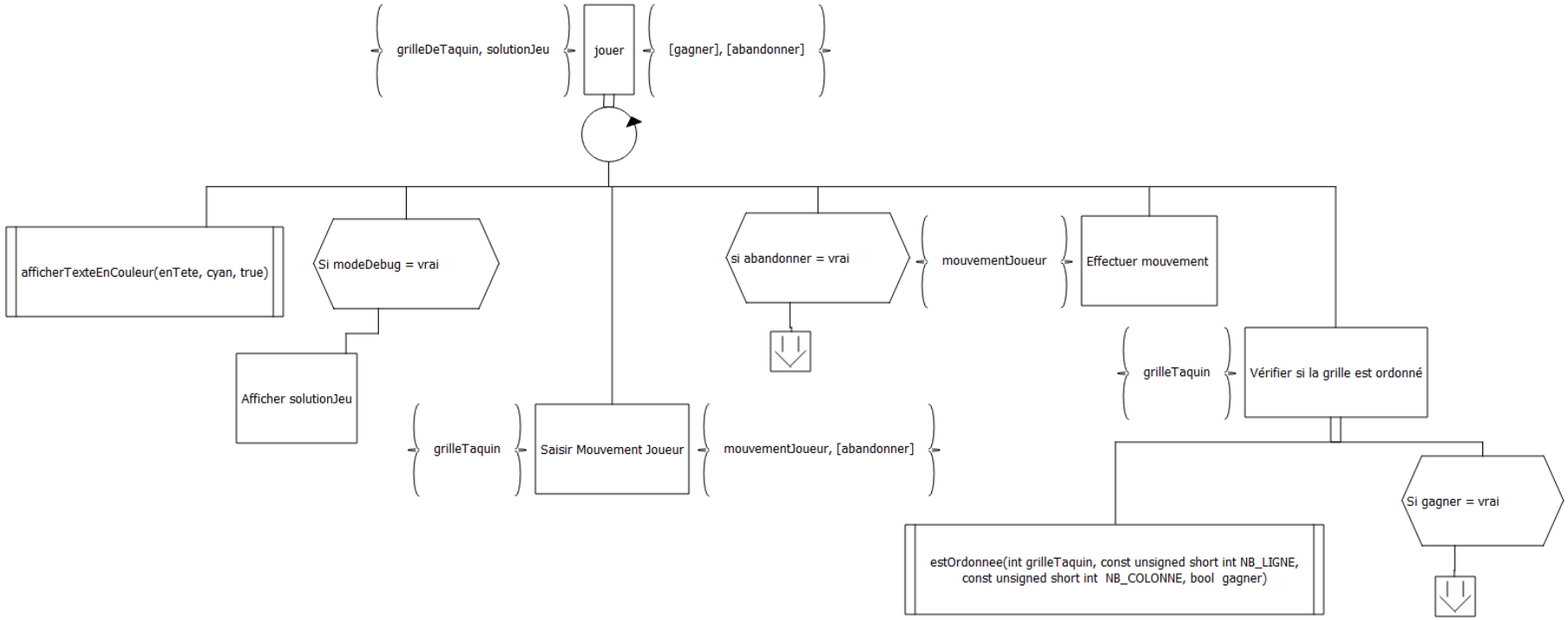
34

NB_LIGNE	Const unsigned short int	Nombre de ligne du tableau
NB_COLONNE	Const unsigned short int	Nombre de colonne du tableau
indiceParcours	Unsigned short int	Indice de parcours des lignes du tableau
indiceParcoursColonne	Unsigned short int	Indice de parcours des colonnesdu tableau

35

SECTION JOUER

Algorithme



36

Description des sous-problèmes

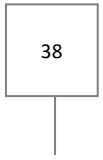
Cet algorithme affiche tout d'abord l'en tête "-Jeu du Taquin-". Si le modeDebug est initialisé alors il affiche à l'écran le mode debug. Il effectue la saisie du joueur, si le joueur décide d'abandonner devient vrai arrête donc la boucle. Sinon il effectue le mouvement du joueur. Pour finir, il regarde si la grille de taquin est triée alors gagner devient vrai et la boucle prend fin

Dictionnaire des variables

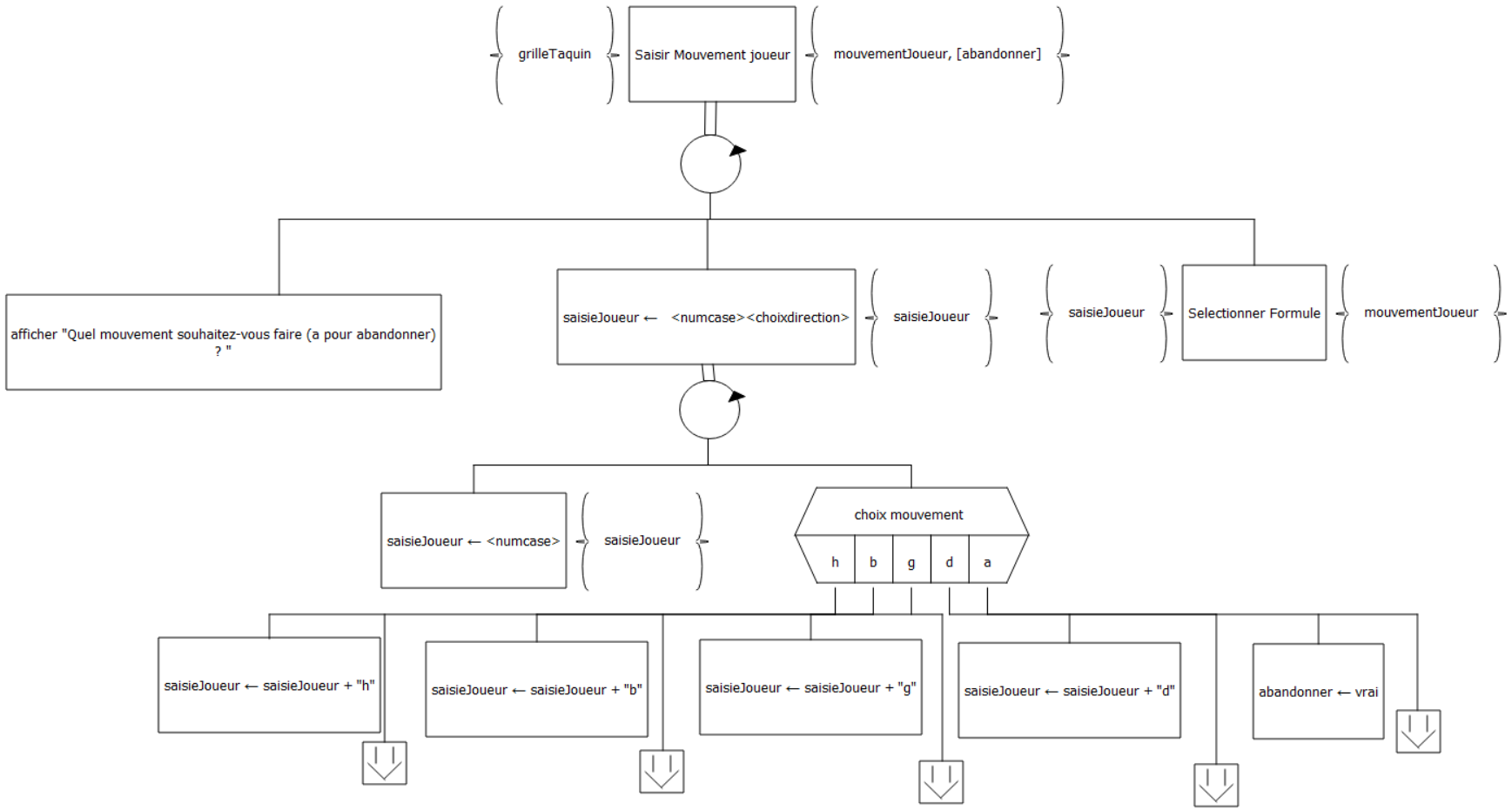
Nom	Type	Signification
grilleTaquin[NB_LIGNE][NB_COLONNE]	int	Tableau d'entiers à deux dimensions de longueur NB_LIGNE et NB_COLONNE
NB_LIGNE	Const unsigned short int	Nombre de ligne du tableau
NB_COLONNE	Const unsigned short int	Nombre de colonne du tableau
modeDebug	Bool	Si vrai alors le mode debug sera affiché à l'écran

37

TITRE ALGO



Algorithme



39

Description des sous-problèmes

Cette algorithme porte sur la saisie du joueur. Le joueur a le choix entre une case et une direction. Ces choix sont affectés à la variable saisieJoueur, sauf si le joueur décide d'abandonner la partie.

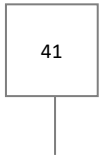
Dictionnaire des variables

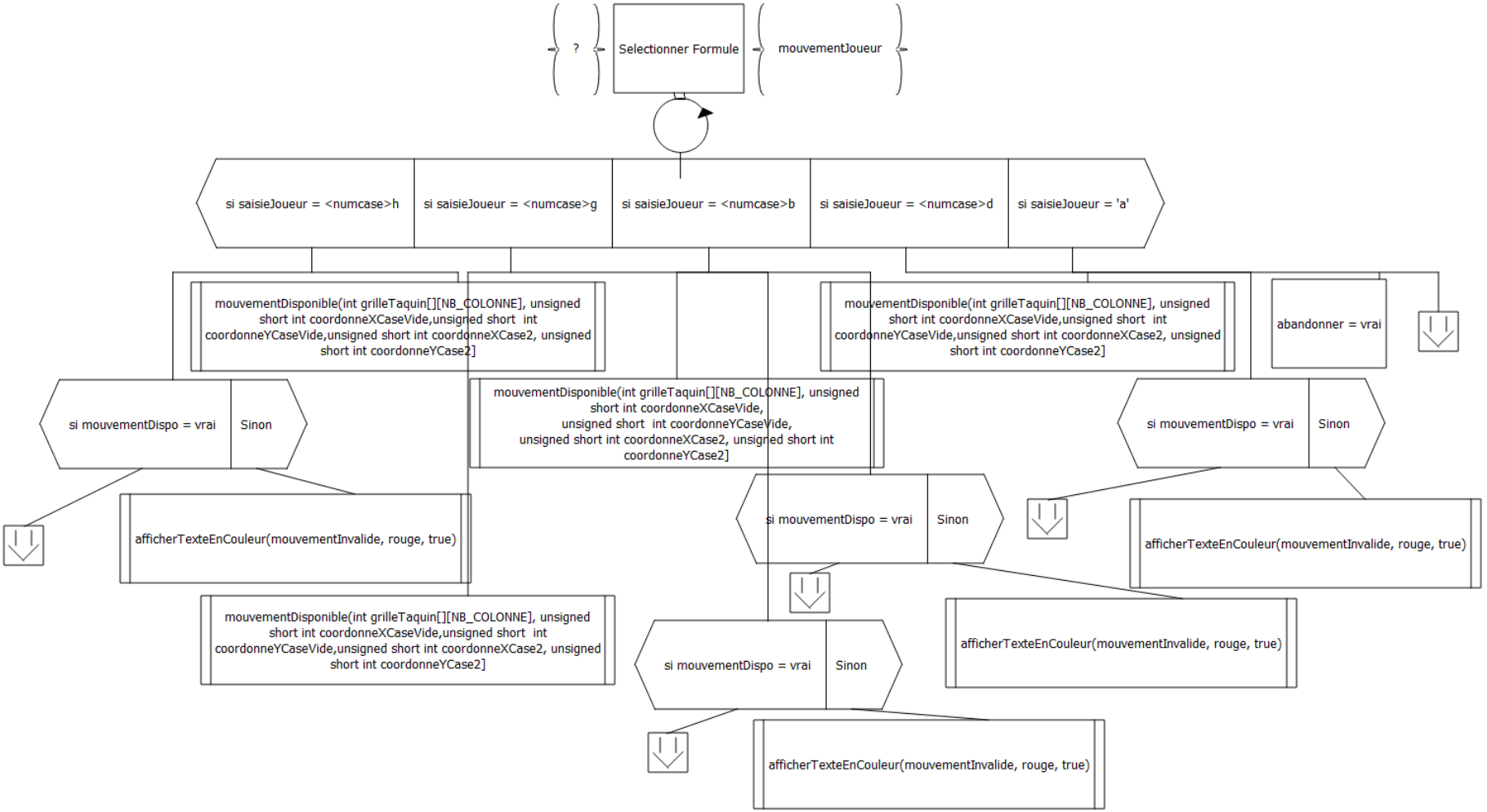
Grilletaquin[][NB_COLONNE]	Int	La grille de Taquin
saisieJoueur	String	La saisie du joueur
abandonner	Bool	Si le joueur décide d'abandonner alors devient vrai, faux sinon

40

TITRE ALGO

Algorithme




?
Selectionner Formule
mouvementJoueur
si saisieJoueur = <numcase>h
si saisieJoueur = <numcase>g
si saisieJoueur = <numcase>b
si saisieJoueur = <numcase>d
si saisieJoueur = 'a'
mouvementDisponible(int grilleTaquin[][NB_COLONNE], unsigned short int coordonneXCaseVide,unsigned short int coordonneYCaseVide,unsigned short int coordonneXCase2, unsigned short int coordonneYCase2]
mouvementDisponible(int grilleTaquin[][NB_COLONNE], unsigned short int coordonneXCaseVide,unsigned short int coordonneYCaseVide,unsigned short int coordonneXCase2, unsigned short int coordonneYCase2]
abandonner = vrai
si mouvementDispo = vrai
Sinon
mouvementDisponible(int grilleTaquin[][NB_COLONNE], unsigned short int coordonneXCaseVide, unsigned short int coordonneYCaseVide, unsigned short int coordonneXCase2, unsigned short int coordonneYCase2]
si mouvementDispo = vrai
Sinon
afficherTexteEnCouleur(mouvementInvalide, rouge, true)
si mouvementDispo = vrai
Sinon
afficherTexteEnCouleur(mouvementInvalide, rouge, true)
mouvementDisponible(int grilleTaquin[][NB_COLONNE], unsigned short int coordonneXCaseVide,unsigned short int coordonneYCaseVide,unsigned short int coordonneXCase2, unsigned short int coordonneYCase2]
si mouvementDispo = vrai
Sinon
afficherTexteEnCouleur(mouvementInvalide, rouge, true)
afficherTexteEnCouleur(mouvementInvalide, rouge, true)

42

Description des sous-problèmes

Cette algorithme sélectionne le choix du joueur. En somme, il vérifie si le mouvement séléctionné par le joueur est valide. Si le mouvement n'est pas valide un message d'erreur s'affiche. Des sous programmes pour les différents mouvements sont utilisé comme mouvementADroiteDispo.

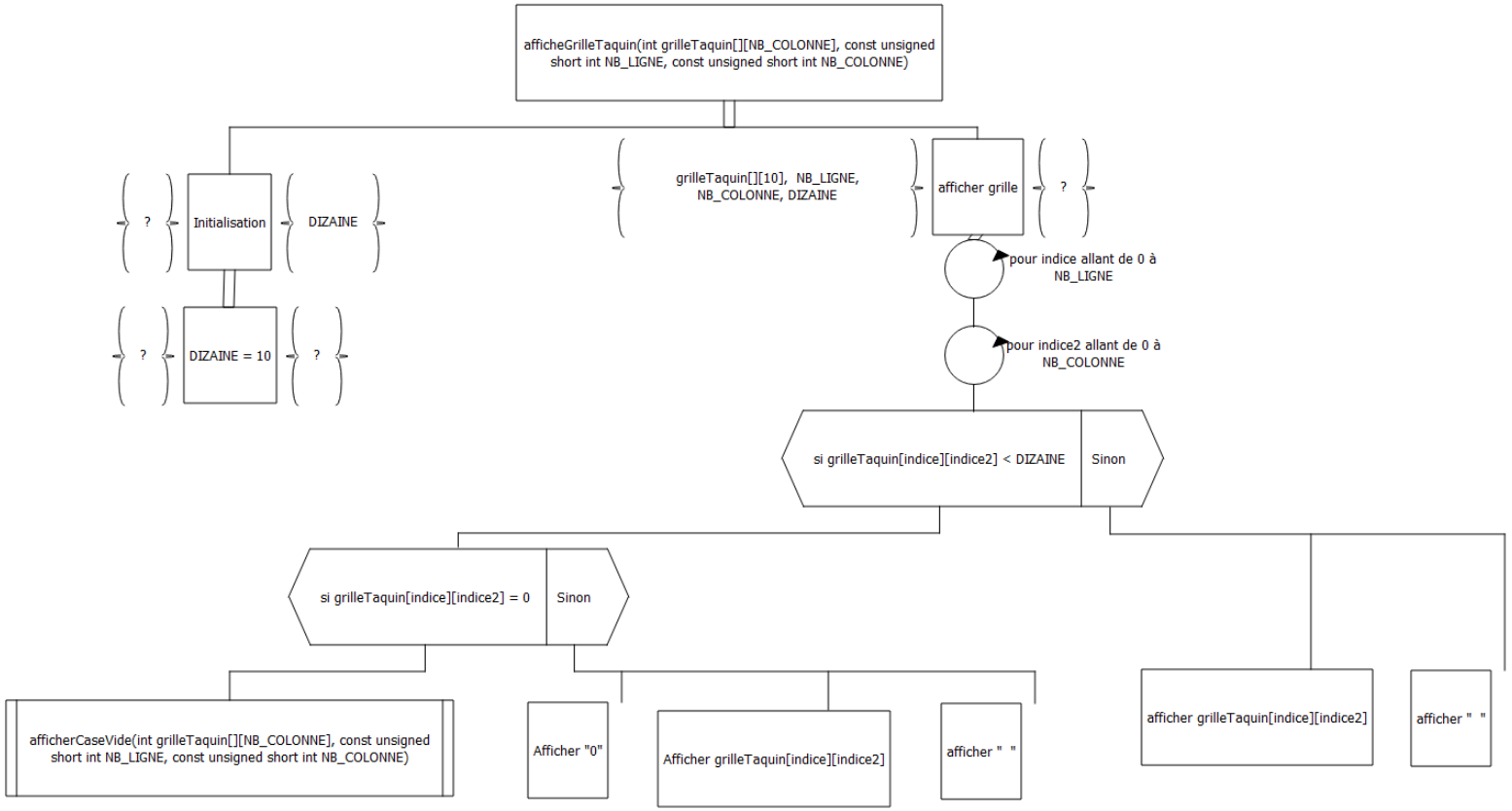
Dictionnaire des variables

mouvementDispo	bool	Si le mouvement du joueur est valide
saisieJoueur	string	La saisie du joueur

43

TITRE ALGO

Algorithme


afficheGrilleTaquin(int grilleTaquin[][NB_COLONNE], const unsigned short int NB_LIGNE, const unsigned short int NB_COLONNE)
?
Initialisation
DIZAINE
?
DIZAINE = 10
?
grilleTaquin[][10], NB_LIGNE, NB_COLONNE, DIZAINE
afficher grille
?
pour indice allant de 0 à NB_LIGNE
pour indice2 allant de 0 à NB_COLONNE
si grilleTaquin[indice][indice2] < DIZAINE
Sinon
si grilleTaquin[indice][indice2] = 0
Sinon
afficherCaseVide(int grilleTaquin[][NB_COLONNE], const unsigned short int NB_LIGNE, const unsigned short int NB_COLONNE)
Afficher "0"
Afficher grilleTaquin[indice][indice2]
afficher " "
afficher grilleTaquin[indice][indice2]
afficher " "

45

Description des sous-problèmes

Cet algorithme affiche la grille de Taquin avec certaines conditions. Si la valeur de la case du tableau est inférieure à 10 alors l'algorithme analyse si la valeur est 0. Si oui, il affiche une case vide sinon il affiche un 0 puis la valeur de la case afin que toutes les lignes du tableau soit aligné. Si la valeur du tableau est supérieure à 10 alors il affiche la valeur de la case du tableau.

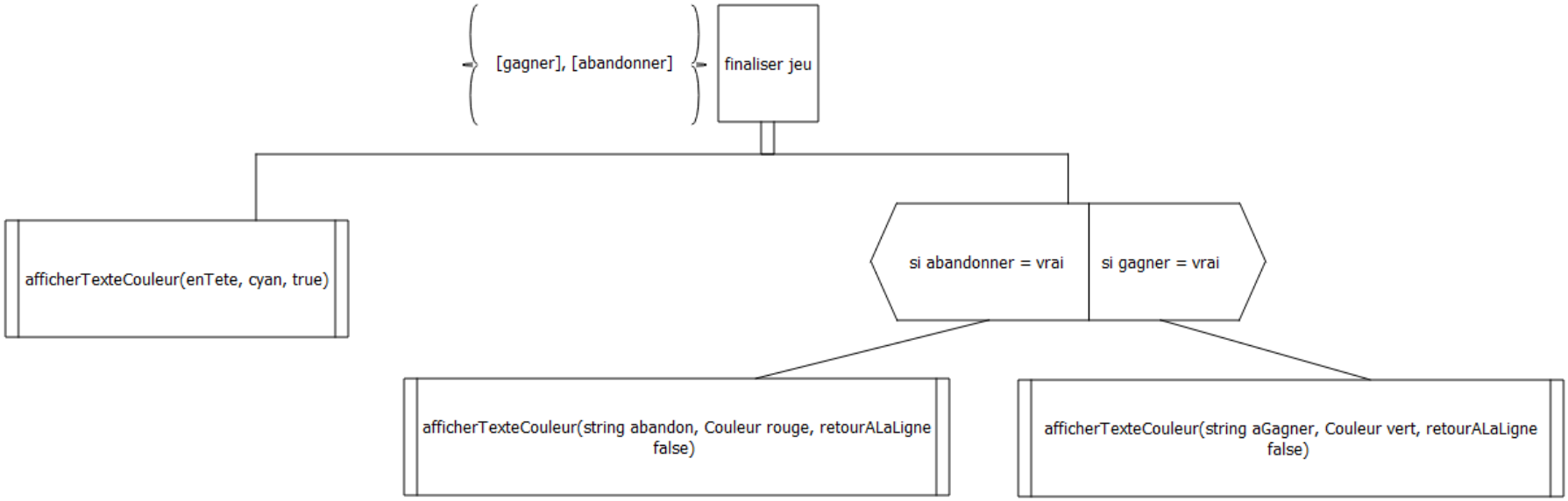
Dictionnaire des variables

Nom	Type	Signification
grilleTaquin[NB_LIGNE][NB_COLONNE]	int	Tableau d'entiers à deux dimensions de longueur NB_LIGNE et NB_COLONNE
NB_LIGNE	Const unsigned short int	Nombre de ligne du tableau
NB_COLONNE	Const unsigned short int	Nombre de colonne du tableau
DIZAINE	Const int	Variable correspondant au nombre 10 qui est untilisé pour des tests

46

TITRE ALGO

Algorithme



47

Description des sous-problèmes

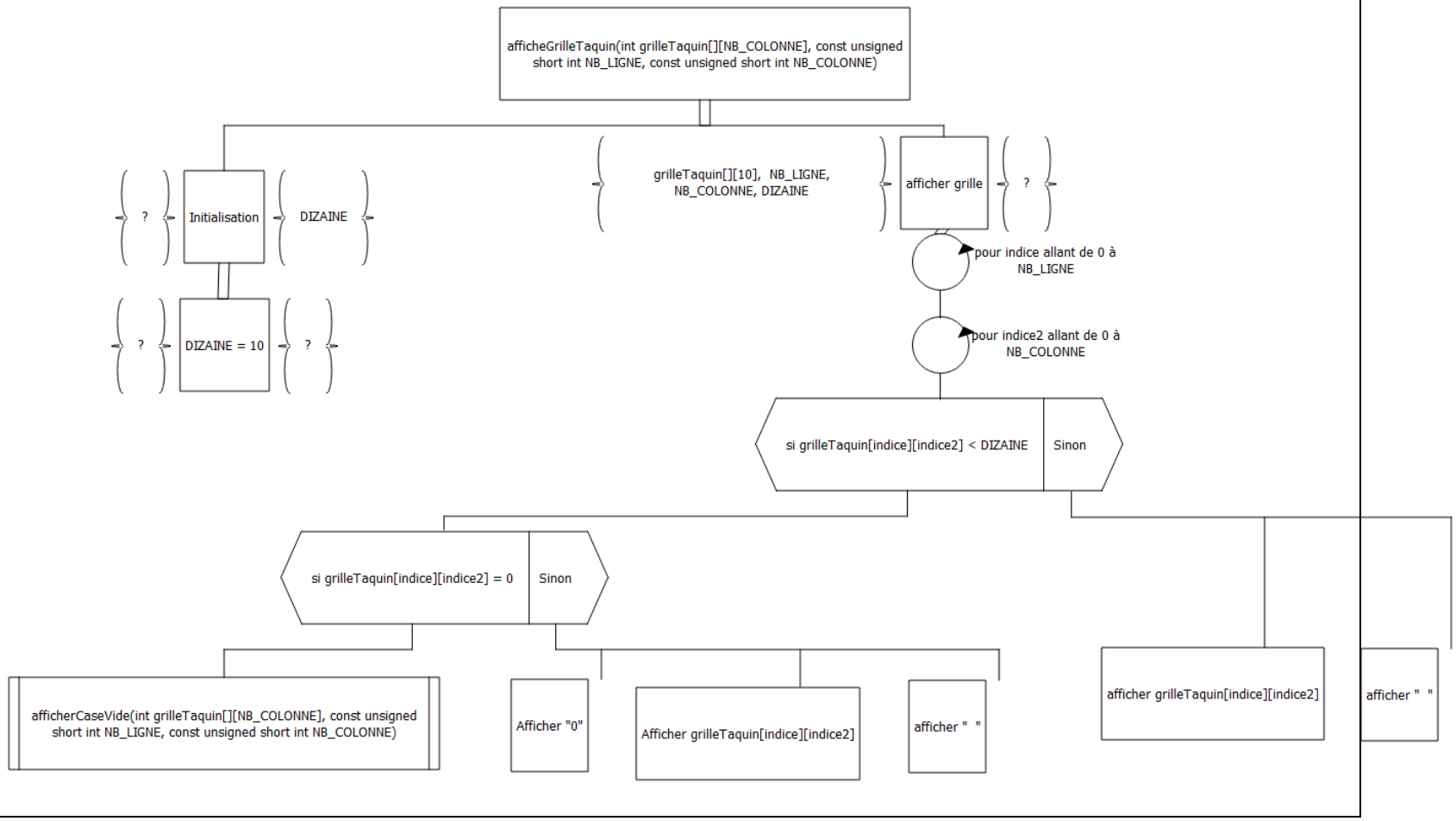
Cet algorithme correspond à la fin de la partie. Si le joueur a gagné et que la grille est ordonnée alors le joueur est félicité. Sinon, il a abandonné alors un message d'abandon s'affiche.

Dictionnaire des variables

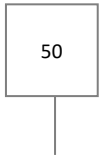
Nom	Type	Signification
gagner	bool	Si le joueur a gagné la partie.
abandonner	Bool	Si le joueur a abandonné la partie
enTete	string	En-tête du jeu
aAbandonner	string	Message si le joueur a abandonné la partie

48

aGagner	String	Message si le joueur a gagné la partie



49


50

A REDIGER UNE FOIS TOUT ÉLÉMENTS

Etat de finalisation :

51

Code source

Main.cpp

```
/* Programme: Jeu course poursuite
But: Faire un jeu pour 2 joueurs basé sur des dé
Date de dernière modif: 23/10/2022
Auteur: Maxime MONTOURO */

#include <iostream>
#include "taquin.h"
#include "game-tools.h"
#include <string>

using namespace std;


int main(void)
{
    //VARIABLES
    int grilleDeTaquin[NB_LIGNE][NB_COLONNE];// Grille de taquin sous forme d'un tableau à deux dimensions de taille
NB_LIGNE et NB_COLONNE.
    unsigned short int indiceXEchange = 0;
    unsigned short int indiceYEchange = 1;
    string caseADeplaceChoixJoueur = "";
    int valeurCaseChoisiPourDeplacement = 0;
    string enTete = " - JEU DU TAQUIN - ";
```


52

```
    string mouvementInvalide = " Mouvement INVALIDE !!! ";
    string aAbandonner = "Perdu par abandon :'-( ";
    string saisieIncorrecte = "Saisie I N C O R R E C T E. Recommencez... (Appuyer sur Entree) ";
    string aGagner = "Felicitations Vous avez gagne la partie ! ";
    string solutionJeu = " La solution est : "; // Solution pour la résolution du Taquin
    char reponseModeDebug;//Reponse pour l'affichage du mode Debug, o si oui et n sinon
    bool modeDebug = false;//Si reponseModeDebug est vrai alors modeDebug est vrai sinon faux, est initialisé a faux
    bool abandonner = false;// Si le joueur décide d'abandonner le jeu au cours de la partie alors abandonner = vrai, est
initialisé à faux
    bool gagner = false;
    string mouvementJoueur;//Le mouvement qu ele joueur souhaite effectuer
    //mouvementJoueurPossible mouvementJoueurChoisi;
    CaseVide positionCaseVide;

    //                                                    INITIALISATION
    //afficherTeteCouleur(enTete) >> écran
    afficherTexteEnCouleur(enTete, cyan, true);

    //Initialisation du mode Debug
    while(true)
    {

       cout << "\n";
        cout << "Voulez-vous activer le mode Debug (o/n) ? ";
        cin >> reponseModeDebug;
        cout << "\n";

        if(reponseModeDebug == 'o')
```


53

```
    {
        modeDebug = true;
        break;
    }
    else if (reponseModeDebug =='n')
    {
        modeDebug = false;
        break;
    }

    afficherTexteEnCouleur(saisieIncorrecte, rouge, false);
    pause();
    effacer();
    afficherTexteEnCouleur(enTete, cyan, true);
}

effacer();
//Generation de la grille de Taquin
genererTaquinNonTrier(grilleDeTaquin, solutionJeu);

//                                              JOUER
//Commencement de la Partie
while(true)
{
    afficherTexteEnCouleur(enTete, cyan, true);
    if(modeDebug == true)
    {
        cout << "\n";
```


54

```
            afficherTexteEnCouleur(solutionJeu, vert,true);
            afficherGrilleTaquin(grilleDeTaquin, NB_LIGNE, NB_COLONNE);
            cout << "\n";
        }
        else
        {
            afficherGrilleTaquin(grilleDeTaquin, NB_LIGNE, NB_COLONNE);
            cout << "\n";
        }

        while(true)
        {
            cout << "\n";
            cout<<"Quel mouvement souhaitez-vous faire (a pour abandonner) ? ";
            cin >> mouvementJoueur;
            convertirSaisieJoueurStringToInt(mouvementJoueur, caseADeplaceChoixJoueur, valeurCaseChoisiPourDeplacement);
            chercherValeurCaseAEchanger(grilleDeTaquin, valeurCaseChoisiPourDeplacement, indiceXEchange, indiceYEchange);
            cout << mouvementJoueur;
            if(mouvementJoueur == "h")
            {

                break;
            }
            else if(mouvementJoueur == "g")
            {
                if(mouvementAGaucheDispo(grilleDeTaquin,
positionCaseVide,valeurCaseChoisiPourDeplacement,  indiceXEchange, indiceYEchange ) == true)
                {
```


55

```
                    faireMouvementGauche(grilleDeTaquin, positionCaseVide, indiceXEchange, indiceYEchange);
                }
                else
                {
                    afficherTexteEnCouleur(saisieIncorrecte, rouge, true);
                }
            }
            else if(mouvementJoueur == "b")
            {
                break;
            }
            else if(mouvementJoueur == "d")
            {
                if(mouvementADroiteDispo(grilleDeTaquin, positionCaseVide,valeurCaseChoisiPourDeplacement,
indiceXEchange, indiceYEchange))
                {
                    faireMouvementDroite(grilleDeTaquin, positionCaseVide, valeurCaseChoisiPourDeplacement,
indiceXEchange, indiceYEchange);
                }
                else
                {
                    afficherTexteEnCouleur(saisieIncorrecte,rouge, true);
                }
            }
            else if(mouvementJoueur == "a")
            {
                abandonner = true;
            }
```


56

```
    else
    {
        cout << "\n";
        afficherTexteEnCouleur(saisieIncorrecte, rouge, false);
        pause();
        effacer();
        afficherTexteEnCouleur(enTete, cyan, true);
        if(modeDebug == true)
        {
            cout<<"\n";
            afficherTexteEnCouleur(solutionJeu, vert,true);
            afficherGrilleTaquin(grilleDeTaquin, NB_LIGNE, NB_COLONNE);
            cout << "\n";
        }
        else
        {
            afficherGrilleTaquin(grilleDeTaquin, NB_LIGNE, NB_COLONNE);
            cout << "\n";
        }
    }
}

if(abandonner == true)
{
    break;
}
```


57

```
        if((estOrdonne(grilleDeTaquin, NB_LIGNE, NB_COLONNE) == true) && (coordonneeFinCaseVide(positionCaseVide,
NB_LIGNE, NB_COLONNE)))
        {
            gagner = true;
            break;
        }
    }

    //                              FIN DE PARTIE

    effacer();
    afficherTexteEnCouleur(enTete, cyan, true);
    afficherGrilleTaquin(grilleDeTaquin, NB_LIGNE, NB_COLONNE);
    if(abandonner == true)
    {
        cout << "\n"<< endl;
        afficherTexteEnCouleur(aAbandonner, rouge, false);
        cout << "\n"<< endl;
        echangerCasesGrille(grilleDeTaquin, positionCaseVide, indiceXEchange, indiceYEchange);
        cout << "\n"<< endl;
        cout << positionCaseVide.coordonneeX << "  " << positionCaseVide.coordonneeY;
    }
    else if(gagner == true)
    {
        cout << "\n" << endl;
        afficherTexteEnCouleur(aGagner, vert, false);
    }
    return 0;
```


58

```
}
```

Taquin.H

```
/**
 * @file taquin.h
 * @author Maxime MONTOURO
 * @brief Fichier d'entête du module taquin
 * @date 2022-19-12
 */
#ifndef TAQUIN_H
#define TAQUIN_H

#include <iostream>
using namespace std;

/*-----------------------------------------------------------------------------------------------------------
                                        VARIABLES GLOBALES
-----------------------------------------------------------------------------------------------------------*/

const unsigned short int NB_LIGNE = 5;// Nombre de ligne d'un tableau
const unsigned short int NB_COLONNE = 5;//Nombre de colonne

/*-----------------------------------------------------------------------------------------------------------
```


59

```
                                        TYPE ENUMERES ET ENRTEGISTREMENTS
-------------------------------------------------------------------------------------------------------------*/

enum mouvementJoueurPossible {h = 1, g = 2, b = 3, d = 4};// Mouvement que peuvent effectuer les différentes cases du
tableau

struct CaseVide
{
    unsigned short int coordonneeX;// Coordonne de la case vide sur l'axe des abscisses du tableau
    unsigned short int coordonneeY;// Coordonne de la case vide sur l'axe des ordonnées du tableau
};

/*-------------------------------------------------------------------------------------------------------------
                                        AFFICHAGE SUR LA GRILLE
-------------------------------------------------------------------------------------------------------------*/

void afficherGrilleTaquin(int grilleTaquin[][NB_COLONNE], const unsigned short int NB_LIGNE, const unsigned short int
NB_COLONNE);
//But: Afficher la grille de Taquin de dimensions NB_LIGNE et NB_COLONNE

void afficherCaseVide(int grilleTaquin[][NB_COLONNE], const unsigned short int NB_LIGNE, const unsigned short int
NB_COLONNE);
//But: Afficher une case vide lorsque la valeur de la case du tableau grilleTaquin de dimensions NB_LIGNE et NB_COLONNE
est 0.

/*-------------------------------------------------------------------------------------------------------------
                                        GENERATION SUR LA GRILLE
-------------------------------------------------------------------------------------------------------------*/
```


60

```
void genererTaquin(int grilleTaquin[][NB_COLONNE], const unsigned short int NB_LIGNE, const unsigned short int
NB_COLONNE);
//But: Génère une grille de taquin ordonnée de dimensions NB_LIGNE et NB_COLONNE

void genererTaquinNonTrier(int grilleTaquin[][NB_COLONNE], string& reponseJeu);
//But: Mélange une grille de taquin  ordonnée  et permet d'avoir accès à la réponse la plus simple reponseJeu si le mode
debug est activé

void initialiserPositionCaseVide(CaseVide& pCaseVide, unsigned short int pAbscisse, unsigned short int pOrdonnee);
//But: Initialise la position de la Case Vide pCaseVide aux coordonnées pAbscisse et pOrdonne de la grille

/*-----------------------------------------------------------------------------------------------------
                                          MODIFICATEURS
-----------------------------------------------------------------------------------------------------*/

void echangerCasesGrille(int grilleTaquin[][NB_COLONNE], CaseVide& pCaseVide,unsigned short int indiceXcase2,unsigned
short int indiceYCase2);
//Echanger les deux cases pCaseVide et case2 dont les coordonnees sont indiceXCase2 et indiceYCase2 d'une grille de
taquin suivant le modèle algorithmique echange case d'un tableau

bool estOrdonne(int grilleTaquin[][NB_COLONNE], const unsigned short int NB_LIGNE, const unsigned short int NB_COLONNE);
//But: Renvoie vrai si le tableau grilleTaquin de longueur NB_LIGNE et NB_COLONNE est trier dans l'ordre croissant (ne
prend pas en compte la case vide), renvoie faux sinon

bool coordonneeFinCaseVide(CaseVide& pCaseVide, const unsigned short int NB_LIGNE, const unsigned short int NB_COLONNE);
//But: Renvoie vrai si la position de la case vide pCaseVide se situe dans l'un des quatre coins de la grille de Taquin
de  dimension NB_LIGNE et NB_COLONNE, faux sinon
```


61

```
void faireMouvementGauche(int grilleTaquin[][NB_COLONNE], CaseVide& pCaseVide,
                          unsigned short int coordonneXCaseAEchanger, unsigned short int coordonneeYCaseAEchanger);
//But: Si le mouvement à gauche est disponible, la procédure échange les cases dudu tableau et change les coordonnées de
la caseVide et de coordonneeXCaseAEchanger et coordonneeYCaseAEchanger

bool mouvementAGaucheDispo(int grilleTaquin[][NB_COLONNE], CaseVide& pCaseVide, int caseChoisiPourDeplacement,  unsigned
short int coordonneeXCase, unsigned short int coordonneeYCase);
//But: Renvoie vrai, si la case a echanger de dimension indiceXCase et indiceYCase est situe à gauche de la case vide
pCaseVide dans le tableau grilleTaquin de dimensions NB_LIGNE et NB_COLONNE

void convertirSaisieJoueurStringToInt(string& saisieJoueur, string& deplacementChoisi, int& caseChoisiPourDeplacement);
//But: convertie la saisie du joueur saisieJoueur en deux le mouvement choisi delacementChoisi et la case choisi
caseChoisiPourDeplacement

void chercherValeurCaseAEchanger(int grilleTaquin[][NB_COLONNE], int caseChoisiPourDeplacement,
                                 unsigned short int& coordonneeXCase, unsigned short int& coordonneeYCase);
//But: Cherche dans la grille de taquin grilleTaquin les coordonnes de la
void faireMouvementDroite(int grilleTaquin[][NB_COLONNE], CaseVide& pCaseVide,int caseChoisiPourDeplacement,  unsigned
short int coordonneXCaseAEchanger, unsigned short int coordonneeYCaseAEchanger);

bool mouvementADroiteDispo(int grilleTaquin[][NB_COLONNE], CaseVide& pCaseVide,int caseChoisiPourDeplacement,  unsigned
short int coordonneeXCase, unsigned short int coordonneeYCase);
//But: Renvoie vrai, si la case a echanger de dimension indiceXCase et indiceYCase est situe à gauche de la case vide
pCaseVide dans le tableau grilleTaquin de dimensions NB_LIGNE et NB_COLONNE

void faireMouvementBas(int grilleTaquin[][NB_COLONNE], CaseVide& pCaseVide, int caseChoisiPourDeplacement, unsigned short
int coordonneeXCaseAEchanger,const unsigned short int coordonneeYCaseAEchanger);
```


62

```
bool mouvementEnBasDispo(int grilleTaquin[][NB_COLONNE], CaseVide& pCaseVide, int caseChoisiPourDeplacement, unsigned
short int coordonneeXCase, unsigned short int coordonneeYCase);

#endif
```

Taquin.cpp

```
#include "taquin.h"
#include "game-tools.h"
#include <iostream>
#include <string>

using namespace std;


/*-----------------------------------------------------------------------------------------------------------
                                        AFFICHAGE SUR LA GRILLE
-----------------------------------------------------------------------------------------------------------*/

void afficherGrilleTaquin(int grilleTaquin[][NB_COLONNE], const unsigned short int NB_LIGNE, const unsigned short int
NB_COLONNE)
{
```


63

```
//VARIABLES
const int DIZAINE = 10;//si valeur du tableau inferieure a 10 alors affiche ' nbr '

//CODE
for(unsigned short int indice = 0; indice < NB_LIGNE; indice++)
{
    cout << "\n";
    for(unsigned short int indice2 = 0; indice2 < NB_COLONNE; indice2++)
    {
        if(grilleTaquin[indice][indice2] < DIZAINE)
        {
            if(grilleTaquin[indice][indice2] == 0)
            {
                afficherCaseVide(grilleTaquin,NB_LIGNE,NB_COLONNE);
            }
            else
            {
                cout <<"0" <<grilleTaquin[indice][indice2] << "  ";
            }
        }
        else
        {
            cout << grilleTaquin[indice][indice2] << "  ";
        }

    }

}
```


64

```
}

void afficherCaseVide(int grilleTaquin[][NB_COLONNE], const unsigned short int NB_LIGNE, const unsigned short int
NB_COLONNE)
{
    for(unsigned short int indiceParcours = 0; indiceParcours < NB_LIGNE; indiceParcours++)
    {
        for(unsigned short int indiceParcoursColonne = 0; indiceParcoursColonne < NB_COLONNE; indiceParcoursColonne++)
        {
            if(grilleTaquin[indiceParcours][indiceParcoursColonne] == 0)
            {
                cout<< "    ";
            }
        }
    }
}

/*-----------------------------------------------------------------------------------------------------------
                                        GENERATION SUR LA GRILLE
-----------------------------------------------------------------------------------------------------------*/

void initialiserPositionCaseVide(CaseVide& pCaseVide, unsigned short int pAbscisse, unsigned short int pOrdonnee)
{
    pCaseVide.coordonneeX = pAbscisse;
    pCaseVide.coordonneeY = pOrdonnee;
}
```


65

```
void genererTaquin(int grilleTaquin[][NB_COLONNE], const unsigned short int NB_LIGNE, const unsigned short int
NB_COLONNE)
{
    //VARIABLE

    int compteur = 0;// Le compteur du tableau initialisé à 0
    CaseVide positionCaseVide;
    unsigned short int coordonneAbscisseCaseVide = 0;
    unsigned short int coordonneOrdonneeCaseVide = 0;

    //CORPS DU SOUS-PROGRAMME
    //Initialisation de la grille de Taquin
    for(unsigned short int indiceParcours = 0; indiceParcours < NB_LIGNE; indiceParcours++)
    {
        for(unsigned short int indice = 0; indice < NB_COLONNE; indice++)
        {
            grilleTaquin[indiceParcours][indice] = compteur;
            compteur++;
        }
    }
    initialiserPositionCaseVide(positionCaseVide, coordonneAbscisseCaseVide, coordonneOrdonneeCaseVide);
}

void genererTaquinNonTrier(int grilleTaquin[][NB_COLONNE], string& reponseJeu)
{
    //VARIABLES
    int mouvementRandom;// nombre de mouvement à faire pour mélanger la grille
    int mouvementRandomTour1;// mouvement à faire au premier tour (controlé pour faire un mouvement valide)
```


66

```
    int nombreTourMelange;//nombre de tour du mélange itéré à chaque tour du parcours
    CaseVide positionCaseVide;
    unsigned short int coordonneeX;//coordonnee X de la case à changer
    unsigned short int coordonneeY;//coordonnee Y de la case à echanger
    string deplacementAFaire;//deplacement à faire (ici il s'agit du mouvement ex: 'g')
    int caseChoisiPourDeplacement = 0;
    int mouvementPrecedent;// Acces au mouvement precedent (1 pour bas, 2 pour droite, 3 pour haut, 4 pour gauche)

    //INTIALIATION
    mouvementRandom = random(5, 15);
    mouvementRandomTour1 = random(1,2);
    nombreTourMelange = 0;
    coordonneeX = 0;//coordonnee X de la case en dessous de la case vide lorsque celle ci n est pas encore triee
    coordonneeY = 1;//coordonnee Y de la case en dessous de la case vide lorsque celle ci n est pas encore triee

    genererTaquin(grilleTaquin, NB_LIGNE, NB_COLONNE);//initialisation de la grille de taquin triée

    for(unsigned short int indiceParcoursMelange = 0; indiceParcoursMelange < mouvementRandom; indiceParcoursMelange++)
    {
        if(nombreTourMelange == 0)
        {
            if(mouvementRandomTour1 == 1)
            {
                faireMouvementBas(grilleTaquin, positionCaseVide, caseChoisiPourDeplacement, coordonneeX, coordonneeY );
                nombreTourMelange++;
                mouvementPrecedent = 1;
                reponseJeu = to_string(grilleTaquin[positionCaseVide.coordonneeX][positionCaseVide.coordonneeY])+
deplacementAFaire + "  ";
```


67

```
            }
            else
            {
                faireMouvementDroite(grilleTaquin, positionCaseVide, caseChoisiPourDeplacement, coordonneeX, coordonneeY
);
                nombreTourMelange++;
                mouvementPrecedent = 2;
                reponseJeu = to_string(grilleTaquin[coordonneeX][coordonneeY])+ deplacementAFaire + "  ";
            }
        }
    }

}

/*-----------------------------------------------------------------------------------------------------
                                        MODIFICATEURS
-----------------------------------------------------------------------------------------------------*/

void echangerCasesGrille(int grilleTaquin[][NB_COLONNE], CaseVide& pCaseVide,unsigned short int indiceXCase2, unsigned
short int indiceYCase2)
{
    //ECHANGE DES CASES DU TABLEAUX
    int copieCase1 = grilleTaquin[pCaseVide.coordonneeX][pCaseVide.coordonneeY];
    grilleTaquin[pCaseVide.coordonneeX][pCaseVide.coordonneeY] = grilleTaquin[indiceXCase2][indiceYCase2];
    grilleTaquin[indiceXCase2][indiceYCase2] = copieCase1;

    //MODIFICATIONS DES COORDONNEES DE LA CASE VIDE
```

```
    pCaseVide.coordonneeX = indiceXCase2;//modification de la coordonnee X de la case vide
    pCaseVide.coordonneeY = indiceYCase2;//modification de la coordonnee Y de la case vide
}

bool estOrdonne(int grilleTaquin[][NB_COLONNE], const unsigned short int NB_LIGNE, const unsigned short int NB_COLONNE)
{
    //VARIABLES
    int compteur = 1;
    bool estVerifier = true;

    //CORPS DU SOUS PROGRAMME

    for(unsigned short int indiceParcoursLigne = 0; indiceParcoursLigne < NB_LIGNE; indiceParcoursLigne++)
    {
        for(unsigned short int indiceParcoursColonne = 0; indiceParcoursColonne < NB_COLONNE; indiceParcoursColonne++)
        {
            if(grilleTaquin[indiceParcoursLigne][indiceParcoursColonne] != 0)
            {
                if(grilleTaquin[indiceParcoursLigne][indiceParcoursColonne] != compteur)
                {
                    estVerifier = false;
                    break;
                }
            }
            compteur++;
        }
    }
```


69

```
    return estVerifier;
}

bool coordonneeFinCaseVide(CaseVide& pCaseVide, const unsigned short int NB_LIGNE, const unsigned short int NB_COLONNE)
{
    //VARIABLE
    bool positionValideFinPartie = false;

    //CORPS DU SOUS PROGRAMME

    if(((pCaseVide.coordonneeX == 0)&&(pCaseVide.coordonneeY == 0)) || ((pCaseVide.coordonneeX ==
0)&&(pCaseVide.coordonneeY==NB_COLONNE)) || ((pCaseVide.coordonneeX == NB_LIGNE)&&(pCaseVide.coordonneeY==0)) ||
((pCaseVide.coordonneeX == NB_LIGNE)&&(pCaseVide.coordonneeY==NB_COLONNE)))
    {
        positionValideFinPartie = true;
    }
    return positionValideFinPartie;
}

void faireMouvementGauche(int grilleTaquin[][NB_COLONNE], CaseVide& pCaseVide, unsigned short int
coordonneeXCaseAEchanger,const unsigned short int coordonneeYCaseAEchanger)
{
    echangerCasesGrille(grilleTaquin, pCaseVide, coordonneeXCaseAEchanger, coordonneeYCaseAEchanger);
}

bool mouvementAGaucheDispo(int grilleTaquin[][NB_COLONNE], CaseVide& pCaseVide, int caseChoisiPourDeplacement, unsigned
short int coordonneeXCase, unsigned short int coordonneeYCase)
{
```


70

```
    //CORPS DU SOUS PROGRAMME

    chercherValeurCaseAEchanger(grilleTaquin, caseChoisiPourDeplacement, coordonneeXCase, coordonneeYCase);

    if((pCaseVide.coordonneeY == (coordonneeYCase-1)) && (grilleTaquin[coordonneeXCase][coordonneeYCase] ==
caseChoisiPourDeplacement))
    {
        return true;
    }
    else
    {
        return false;
    }
}

void convertirSaisieJoueurStringToInt(string& saisieJoueur,string& deplacementChoisi, int& caseChoisiPourDeplacement)
{

    //CORPS DU SOUS PROGRAMME

    for(unsigned short int indice = 0; indice < saisieJoueur.size(); indice++)
    {
        if(indice != (saisieJoueur.size()-1))
        {
            deplacementChoisi += saisieJoueur[indice];
            caseChoisiPourDeplacement = stoi(deplacementChoisi);
        }
```


71

```
        else
        {
            deplacementChoisi += saisieJoueur[indice];
        }
    }
}

void chercherValeurCaseAEchanger(int grilleTaquin[][NB_COLONNE], int caseChoisiPourDeplacement, unsigned short int&
coordonneeXCase, unsigned short int& coordonneeYCase)
{




    for(unsigned short int indiceParcours1 = 0; indiceParcours1 < NB_LIGNE; indiceParcours1++)
    {
        for(unsigned short int indiceParcours2 = 0; indiceParcours1 < NB_COLONNE; indiceParcours2++)
        {
            if(grilleTaquin[indiceParcours1][indiceParcours2] == caseChoisiPourDeplacement)
            {
                coordonneeXCase = indiceParcours1;
                coordonneeYCase = indiceParcours2;
            }
        }
    }
}
```


72

```
void faireMouvementDroite(int grilleTaquin[][NB_COLONNE], CaseVide& pCaseVide, int caseChoisiPourDeplacement, unsigned
short int coordonneeXCaseAEchanger, unsigned short int coordonneeYCaseAEchanger)
{
    if(mouvementADroiteDispo(grilleTaquin, pCaseVide, caseChoisiPourDeplacement, coordonneeXCaseAEchanger,
coordonneeYCaseAEchanger) == true)
    {
        echangerCasesGrille(grilleTaquin, pCaseVide, coordonneeXCaseAEchanger, coordonneeYCaseAEchanger);
    }
}

bool mouvementADroiteDispo(int grilleTaquin[][NB_COLONNE], CaseVide& pCaseVide, int caseChoisiPourDeplacement, unsigned
short int coordonneeXCase, unsigned short int coordonneeYCase)
{
    //CORPS SOUS PROGRAMMES
    chercherValeurCaseAEchanger(grilleTaquin, caseChoisiPourDeplacement, coordonneeXCase, coordonneeYCase);

    if((coordonneeXCase-1 == pCaseVide.coordonneeX) && (coordonneeYCase == pCaseVide.coordonneeY) &&
(grilleTaquin[coordonneeXCase][coordonneeYCase] == caseChoisiPourDeplacement))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```


73

```
void faireMouvementBas(int grilleTaquin[][NB_COLONNE], CaseVide& pCaseVide, int caseChoisiPourDeplacement, unsigned short
int coordonneeXCaseAEchanger,const unsigned short int coordonneeYCaseAEchanger)
{
    if(mouvementEnBasDispo(grilleTaquin, pCaseVide, caseChoisiPourDeplacement, coordonneeXCaseAEchanger,
coordonneeYCaseAEchanger) == true)
    {
        echangerCasesGrille(grilleTaquin, pCaseVide, coordonneeXCaseAEchanger, coordonneeYCaseAEchanger);
    }
}

bool mouvementEnBasDispo(int grilleTaquin[][NB_COLONNE], CaseVide& pCaseVide, int caseChoisiPourDeplacement,  unsigned
short int coordonneeXCase, unsigned short int coordonneeYCase)
{
    chercherValeurCaseAEchanger(grilleTaquin,caseChoisiPourDeplacement, coordonneeXCase, coordonneeYCase);
    if((coordonneeXCase == pCaseVide.coordonneeX) && (coordonneeYCase == pCaseVide.coordonneeY-1) &&
(grilleTaquin[coordonneeXCase][coordonneeYCase] == caseChoisiPourDeplacement))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```


74