INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN NORMANDIE

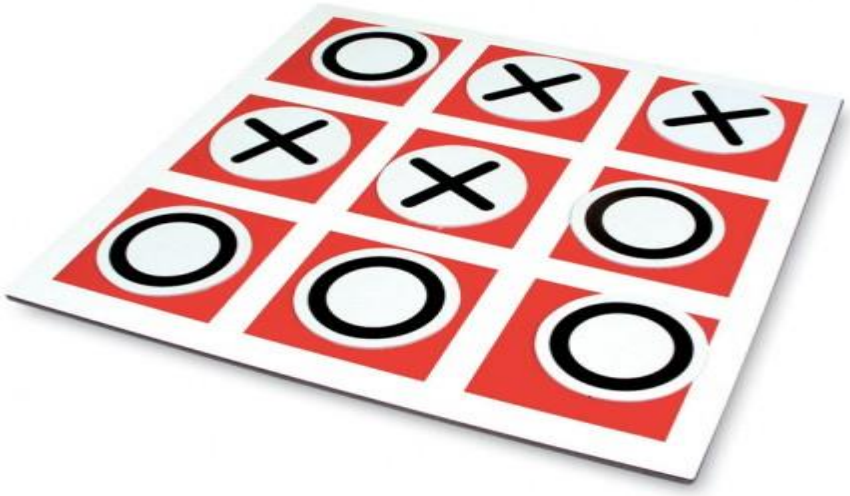


Figure 1 : Jeu Morpion

Rapport De projet : Jeu Morpion

-Département EM-S.I.C-

KOSSIR elmehdi

BENKHALIL hamza

BAAHMED abdessamad

Sommaire

I-Introduction:

Le but de notre projet est d'enregistrer les coups des joueurs d'une partie de Morpion, afin de pouvoir accéder à un historique nous permettant de déterminer les performances des différents joueurs et désigner les vainqueurs de chaque partie.

Notre première tâche a été de développer au brouillon l'analyse descendante du programme, ceci nous a permis d'identifier les modules et les procédures utiles à la conception de notre programme.

II-Règles du jeu

Notre jeu du Morpion s'appuie sur une grille 3x3 et chaque partie à deux issues:

- Grille complète mais pas de vainqueurs.
- Grille incomplète mais un des joueurs a pu aligner 3 pions et donc ce dernier remporte la partie.

Ainsi nous avons aussi décidé que le vainqueur du jeu serait le joueur avec les meilleures performances sur un total de partie dont le nombre est désigné par l'utilisateur.

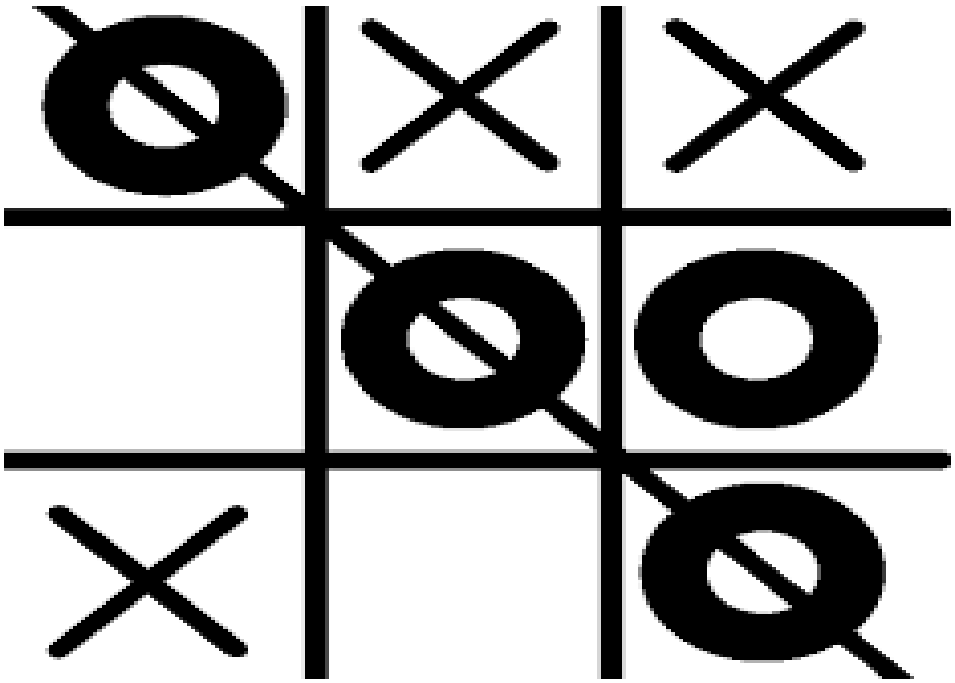
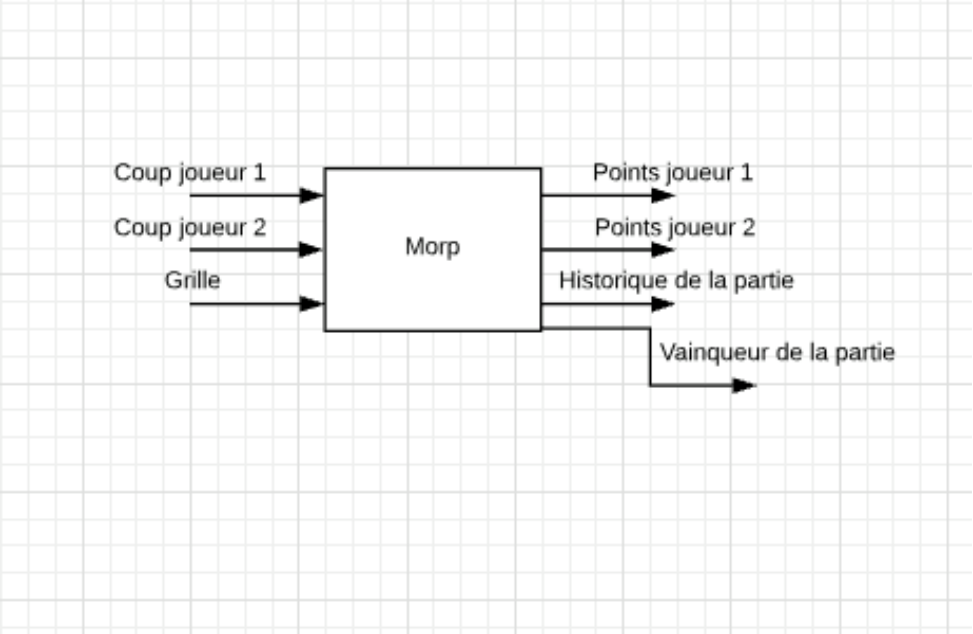


Figure2 : Exemple d'alignement des pions permettant à un des joueurs de réclamer la victoire.

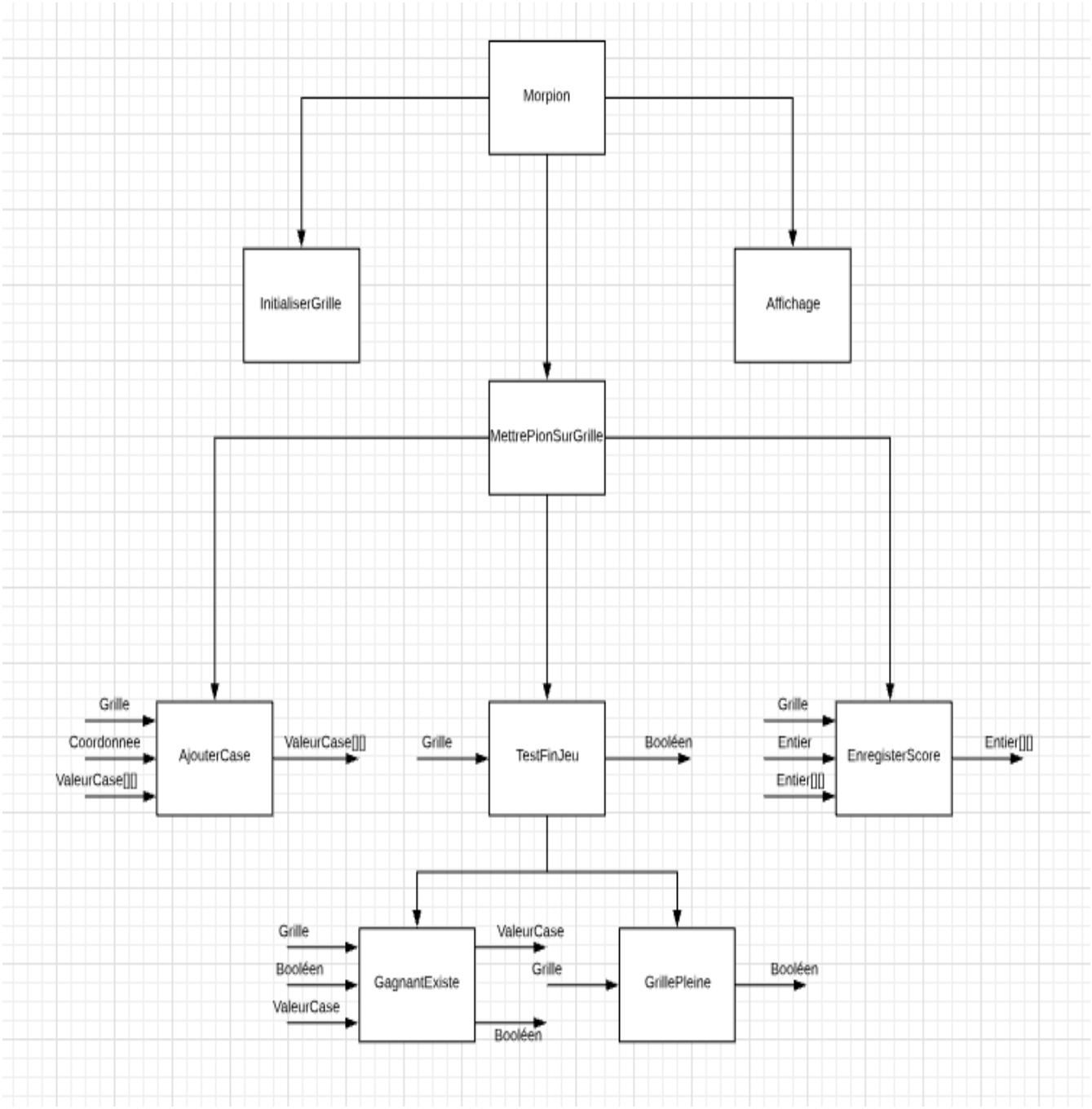
III-MORP :

Acronyme choisi :MORP

L'acronyme que nous avons choisi pour décrire le jeu de morpion est : MORP. En entrée un des joueurs choisit une case ou mettre son pion , en contre partie le jeu enregistre les coups des différents joueurs afin d'en garder une trace au niveau de l'historique , cela nous permet de compter les points des joueurs afin d'en déterminer le gagnant.



IV-Analyse descendante



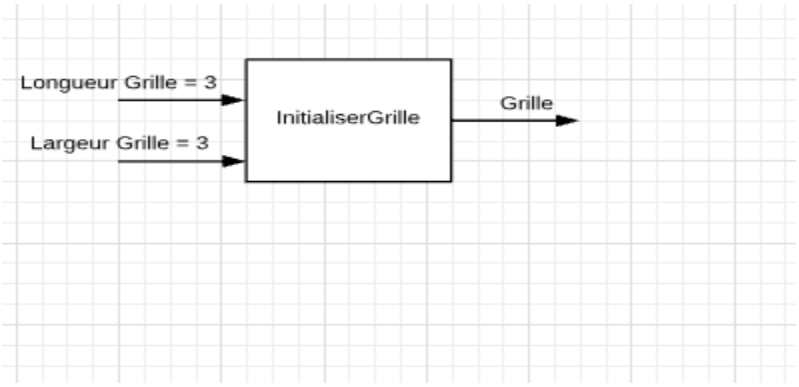
V- Modules :

Grace à l'analyse descendante, nous avons mis en exergue 3 modules : InitialiserGrille ;metUnPionSurLaGrille ;Affichage.

1-InitialiserGrille

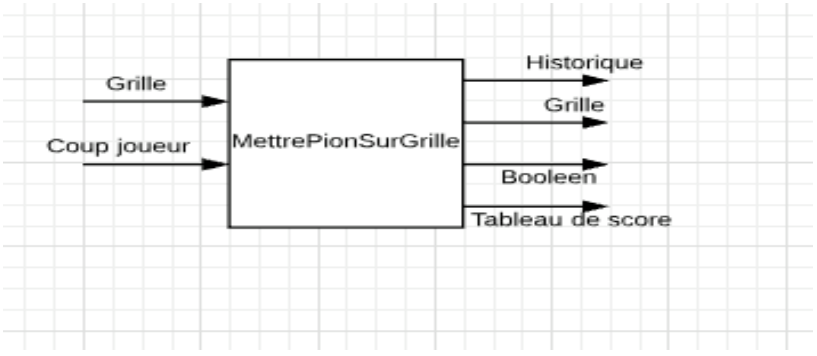
Ce module intervient au début de partie. Il sert à initialiser la grille du morpion à vide.

On a travaillé avec une grille de taille 3*3.



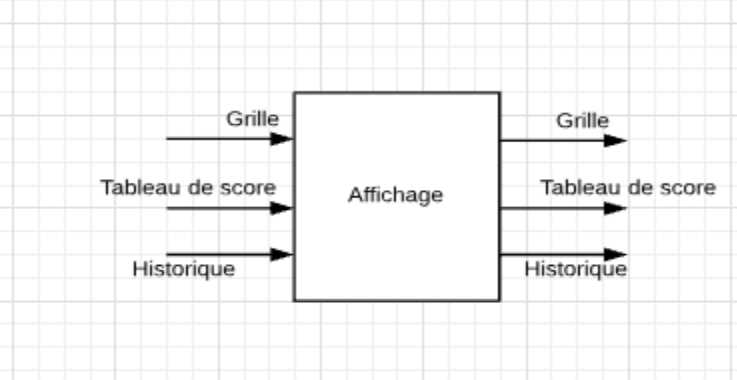
2-metUnpionSurLaGrille

Saisit les coordonnées du nouveau pion à mettre sur la grille, si les coordonnées sont en dehors de la grille ou si la case possède déjà un pion la saisie est refusée sinon on ajoute au tableau d'historique dont les dimensions représentent les coordonnées des cases le pion, on teste si le jeu est fini, si c'est le cas on enregistre le score dans le tableau de score.



3-Affichage :

Module permettant d'afficher la grille du jeu ainsi que le tableau des scores et l'historique.



VI-Algorithme du jeu :

Pseudo-Code

Conformément à notre analyse descendante, d'abord on récupère le nombre de partie entré par l'utilisateur, ensuite, on initialise la grille de taille 3*3, on ressort l'historique et le tableau de score pour chaque partie terminée avec la procédure metUnPionSurLaGrille, et on affiche la grille ainsi que le tableau de score et l'historique.

Procedure main ()
Déclaration :
nbPartie : Naturel
tabScore : Tableau[2][nbPartie+1] de Naturel(tableau qui contiendra les scores dont la taille dépend du nombre de partie entrée par l'utilisateur)
Historique : Tableau[3][3] de ValeurCase (voir TAD Case)(tableau qui contiendra l'historique de chaque partie)

Début
Ecrire(Veuillez entrer le nombre de partie jouée)
Lire(nbPartie)
grille<-initialiserGrille ()
metUnPionSurLaGrille(grille,coordonnece,tabScore,Historique)
afficher(grille,tabScore,Histrique)
Fin

VII-Procédure / fonctions -module :metUnpionSurLaGrille

Plusieurs fonctions et procédures interviennent dans ce module :

- **ajouterCase** : procédure qui permet d'ajouter une case ainsi que sa valeur dans la grille à l'historique afin de garder une trace des différents coups des joueurs de la partie.
  Entrée : Grille, coordonnée
  Entrée/Sortie : Tableau[3][3] de ValeurCase

- **TestFinJeu** : On teste si le jeu est finie pour ça on utilise deux fonctions :
  -GagnantExiste : procédure qui teste si l'un des joueurs a gagnés (ligne, colonne ou diagonale remplit de pions semblables).Dans ce cas la partie est finie.
  Entrée : Grille
  Entrée/Sortie : Booleen
  Sortie : ValeurCase

  -**GrillePleine :**fonction qui retourne un booleenS'il n y'a pas de gagnant, teste que la grille n'est pas pleine si elle est pleine elle retourne true ça veut dire qu'aucun joueur n'a gagné et que la partie est finie.
  Entrée : Grille

- **EnregistrerScore** :procédure qui remplit le tableau de score dont la première ligne représente le premier joueur (soit ROND ,CROIX) et la deuxième ligne le deuxième joueur et dont les colonnes représentent les numéros de parties jouées.
  La dernière colonne pour chaque joueur contient le score finale qui est la somme des scores des parties jouées.
  Entrée : Grille, Naturel
  Entrée/Sortie : tableau[2][nbPartie+1] de Naturel

VIII-TAD utilisé Dans Le module :

TAD CASE:

typeenumValeurCase = [VIDE,ROND,CROIX]

Nom : Case

Utilise :ValeurCase, Booleen

Opération :

Case :ValeurCase* Booleen→ case
obtenirValeur : case → ValeurCase
obtenirBooleen : case → Booleen
sontEgales case*case → Booleen

Axiomes :

obtenirValeur (case (v,bool)) = v
obtenirBooleen (case (v,bool)) = bool
sontEgales (case (v,bool), case (v,bool))

a)Conception Détaillée:

-Définition du type Case :

Type Case = structure
Valeur : ValeurCase
estOccupee : booleen

-Fonctions et procédures du TAD Case et issue de notre conception.

Fonction initialisationCase(Valeur : ValeurCase, estOccupee : Booleen) : Case
Fonction obtenirValeur(uneCase : Case) : ValeurCase
FonctionestOccupee (uneCase : Case) : Booleen
FonctionsontEgales(case1 : Case, case2 : Case) : Booleen

b)Pseudo Code :

FonctioninitialisationCase(Valeur :ValeurCase, estOccupee: Booleen) : Case
Debut
uneCase : Case
uneCase.Valeur<- Valeur
uneCase.estOccupee<- estOccupee

```
        retourneruneCase

fonctionobtenirValeur ( uneCase : Case) : ValeurCase
Debut
        retourneruneCase.Valeur
Fin


fonctionestOccupee (uneCase : Case) : Booleen
Debut
        retourneruneCase.estOccupee
Fin

FonctionsontEgales( case1 : Case, case2 : Case ) : Booleen
Debut
        si ( (uneCase1.Valeur = uneCase2.Valeur)et (uneCase1.estOccupee =
uneCase2.estOccupee)) alors
                retourner Vrai
        sinon
                retourner Faux
        finSi
Fin
```

TAD GRILLE:

Nom : Grille
Utilise : Coordonnee, Case
Opération :
- Grille : → Grille
- ajouterCase : Grille*Case*Coordonee → Grille
- obtenirCase : Grille * Coordonee → Case
- changementDEtat:Grille*Coordonnee→ Grille

Axiomes :
- obtenirCase (ajouterCase(G ,c, xy),xy) = c

obtenirBooleen(obtenirCase(changementDEtat(ajouterCase(G,c,xy),xy))=non obtenirBooleen(c)

a)Conception Détaillée:

-Définition du type Case :

Type Grille = tableaux[1..3][1..3] de case

-Fonctions et procédures du TAD Grille et issue de notre conception.

Fonction Grille ():Grille
Fonction obtenirCase(grille:Grille, coordonnee:Coordonnee):Case
Procedure changementDEtat (E/S: uneGrille:Grille, E: coordonnee:Coordonnee)
Procedure modifierCase(grille:Grille, Case uneCase,coordonnee:Coordonnee)

b) Pseudo Code :

-Fonction Grille ():Grille
Declaration:
grille : Grille
i,j : Entier
Debut
  pour i=1 à 3 faire
    pour j=1 à 3 faire
      grille[i][j] <- case(VIDE,faux)
    finPour
  finPour
  retourner(grille)
Fin

-Fonction obtenirCase(grille:Grille, coordonnee):Case

Debut
  retourner(grille[coordonnee.x][coordonee.y]);
Fin

-Procedure modifierCase(grille:Grille, Case uneCase,coordonnee:Coordonnee)
//ajouterCase du TAD mais modifié pour être conforme à notre conception

Debut
grille[coordonnee.x][coordonnee.y]<-uneCase
Fin

-ProcedurechangementDEtat (E/S: grille : Grille, E: coordonnee:Coordonnee)

Debut
grille[coordonnee.x][coordonnee.y].estOccupee<-
non(grille[coordonnee.x][coordonnee.y].estOccupee)
Fin

IX-Tests Unitaire

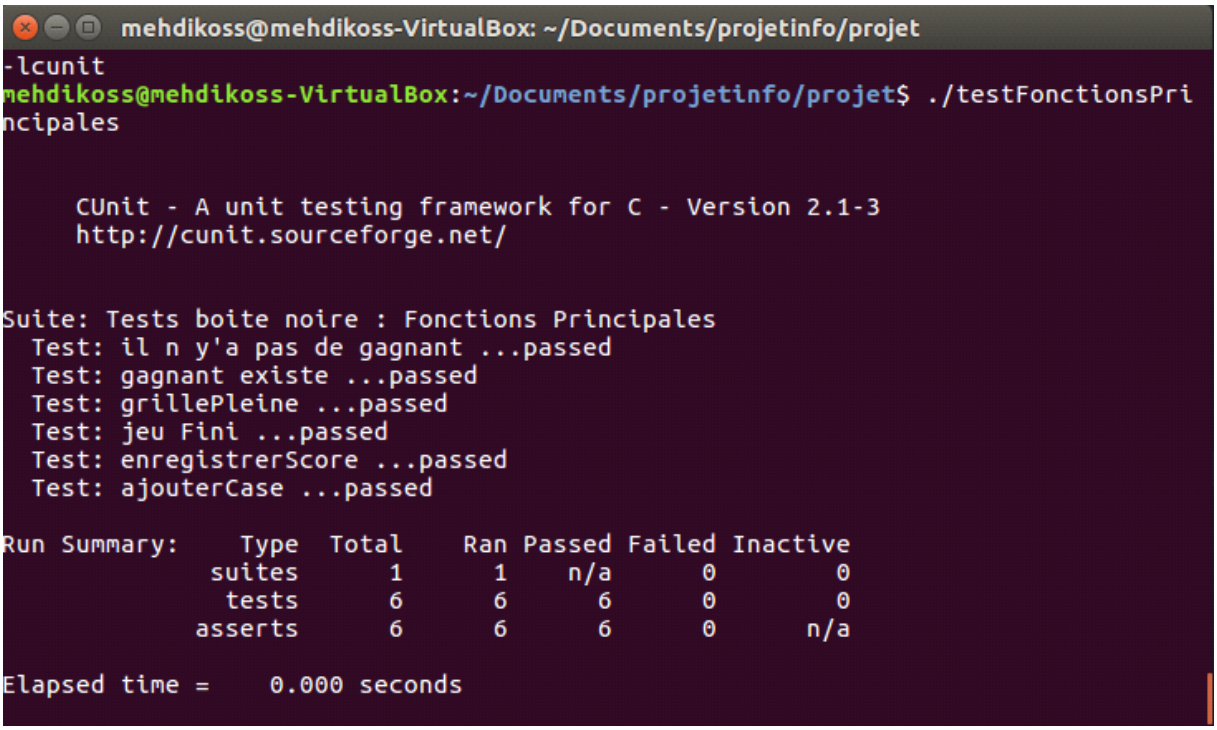
-Test TAD Case:

```
mehdikoss@mehdikoss-VirtualBox: ~/Documents/projetinfo/projet
mehdikoss@mehdikoss-VirtualBox:~/Documents/projetinfo/projet$ gcc -c -Wall Grill
e.c
mehdikoss@mehdikoss-VirtualBox:~/Documents/projetinfo/projet$ ./testCase


     CUnit - A unit testing framework for C - Version 2.1-3
     http://cunit.sourceforge.net/


Suite: Tests boite noire : Case
  Test: obtenir Valeur ...passed
  Test: est deja occupee ...passed
  Test: sont egale ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests      3      3      3      0        0
             asserts      3      3      3      0      n/a

Elapsed time =    0.000 seconds


mehdikoss@mehdikoss-VirtualBox:~/Documents/projetinfo/projet$
```

-Test TAD Grille:

```
mehdikoss@mehdikoss-VirtualBox: ~/Documents/projetinfo/projet


mehdikoss@mehdikoss-VirtualBox:~/Documents/projetinfo/projet$ ./testGrille


     CUnit - A unit testing framework for C - Version 2.1-3
     http://cunit.sourceforge.net/


Suite: Tests Grille
  Test:  Grille ...passed
  Test: Obtenir case ...passed
  Test: ChangementDEtat ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests      3      3      3      0        0
             asserts      3      3      3      0      n/a

Elapsed time =    0.000 seconds


mehdikoss@mehdikoss-VirtualBox:~/Documents/projetinfo/projet$
```

-Test des fonctions/procedures du module : 12



Voir l'archive joint avec le rapport pour les codes complets des tests ainsi que des procédures/fonctions.

X-Conclusion

Ce projet nous a permis de mettre en œuvreun bon nombre de notions vues en cours, nous avons aussi pris conscience de l'importance de l'analyse en vue du développement d'un programme efficace et fiable. Faute de temps nous n'avons pu explorer tous les modules, par contre nous avons choisis celui avec le plus de procédures/fonctions à développer afin de progresser en programmation C, mais aussi pour nous permettre de développer certains reflexes algorithmiques et nous améliorer au niveau des phases d'analyse et de décomposition d'un problème en plusieurs modules et parties. Enfin nous pensons que ce projet nous fût bénéfique en nous permettant de nous éloigner de nos cours théoriques vers quelque chose d'applicatif.