**Projet informatique C# 2016 -2017**

**ETUDIANT : LE Trung Hieu**

**PEIP 2 –POLYTECH ORLEANS**

**C# : Création de jeux Vidéo – Le Bailli**

**MAI 2017**

**Encadrant : M. LECONGE Rémy**

**Sujet personnalisé :** Création du jeu « Le Bailli » en langage C# en utilisant le WINFORM et son graphique.

**Remerciements :**

Je remercie Monsieur LECONGE Remy pour tous ses aides et ses conseils pendant tout le long du projet.

Table des matières

[1. Conceptualisation du Projet 7](#_Toc484100520)

[1.1 Mots clefs et informations pratiques 7](#_Toc484100521)

[1.2 L’Objectif du Projet 8](#_Toc484100522)

[1.3 Calendrier prévisionnel et le calendrier réel 9](#_Toc484100523)

[1.4 La choix : La description structure POO du programme. 12](#_Toc484100524)

[2. Réalisation du projet. 20](#_Toc484100525)

[2.1 Gestions de l’interfaces graphiques et structure des données 20](#_Toc484100526)

[2.1.1 Stockage les positions des pièces, initialise le graphique, les premiers pas de construction 20](#_Toc484100527)

[2.1.2 Initialiser le damier et l’interface graphiques. 21](#_Toc484100528)

[2.1.3 Mis à jour les mouvements des pièces, appliquer la règle du jeu dans le programme, création de la partie 2 joueurs 24](#_Toc484100529)

[2.2 Créations de L’IA 25](#_Toc484100530)

[2.2.1 Transforme la structure d’affichage pour l’IA. 25](#_Toc484100531)

[2.2.2 Réalisations des algorithmes aléatoires contrôlées pour l’ordinateur 26](#_Toc484100532)

[3. Déroule du projet 27](#_Toc484100533)

[3.1. Le travail réalisé 27](#_Toc484100534)

[3.2 La validation de code 27](#_Toc484100535)

[3.3 Les difficulté rencontré 27](#_Toc484100536)

[3.4 Etat final du projet 27](#_Toc484100537)

[3.5 Cahier des charges pour les éventuelles successeurs. 28](#_Toc484100538)

[Conclusion 29](#_Toc484100539)

Table des illustrations

[Figure 1 :position initiale 8](#_Toc484103689)

[Figure 2 Position gagnant d'équipe noire Figure 3 Match nul 8](#_Toc484103690)

[Figure 4 Structure Général 13](#_Toc484103691)

[Figure 5 Le menu 14](#_Toc484103692)

[Figure 6 Espace de jeux 16](#_Toc484103693)

[Figure 7 Message gagnant 16](#_Toc484103694)

[Figure 8 Menu Principal du Programme 21](#_Toc484103695)

[Figure 9 Ressource des pièces 22](#_Toc484103696)

[Figure 10 Retourner la PieceType 22](#_Toc484103697)

[Figure 11 Code pour dessiner et placer les pièces 22](#_Toc484103698)

[Figure 12 Interface Graphique principale 23](#_Toc484103699)

[Figure 13 Schéma mécanisme du jeux, gestion graphique 24](#_Toc484103700)

[Figure 14 Schéma logique de l'IA 26](#_Toc484103701)

**Introduction :**

En Moyen-âge, quand les ordinateurs ne sont pas encore apparus, l’internet et la télévision ne sont pas encore existées. À part les travaux quotidiens, l’Homme a besoin une forme de divertissement pour s’amuser et se détendre. Les jeux de stratégie telles que les échecs, le trictrac, … sont très appréciés spécialement non seulement par des nobles mais aussi par le peuple en Europe.

Avec le développement de la technologie, les ordinateurs sont apparus. Par conséquence, les jeux de stratégie du Moyen Age peuvent être simulés sur les ordinateurs. A partir de cette époque, une nouvelle notion apparue, qui est l’IA (Intelligence Artificielle) et la virtualisation. En effet, le développement de la Science Informatique et le processeur d’ordinateur, l’ordinateur peut devenir un très bon joueur et même dépasser l’Homme dans certains domaines. Aujourd’hui, cette hypothèse devient de plus en plus une réalité, en effet, en 2016, L’IA Alpha Go d’entreprise Google a battu un champion de Jeu de Go. Donc, nous rapprochons de plus en plus cette réalité.

Inspirée de la technologie Simulation, de l’Intelligent Artificielle et des Jeux du Moyen Age. Le but du projet est de construire un concept virtuel de ce jeu et de conceptualiser une IA pour ce jeu. Dans ce rapport, nous allons tout d’abord analyser et construire un raisonnement pour résoudre ce problème. Puis ensuite, nous voyons quelles sont les méthodes qui permettent de construire le programme. Puis nous terminerons avec le bilan du projet et des difficultés rencontrées pendant la période de développement.

# Conceptualisation du Projet

## Mots clefs et informations pratiques

**Mots clés :**

Projet Informatique, C#, Graphique, Algorithme, Réseau, Intelligent Artificiel.

**Information pratiques :**

* Elève participant au projet : LE Trung Hieu, PEIP 2 à Polytech Orléans.
* Tuteur du projet : M. LECONGE Remy, Enseignant Chercheur à Polytech Orléans.
* Nombre de jours pour réaliser le projet : 7 jours
* Période du projet : du lundi 22 mai 2017 au 2 mai 2017
* Echéancier de réalisation :
  + 30 mai 2017 à 18h (le développement de nouvelles fonctionnalités)
  + 1 Juin 2017 à 20h (l’amélioration et fixer les bugs)
* Date de la soutenance : 2 Juin 2017
* Langage de programmation : C#
* Compilateur utilisées : Microsoft Visual Studio 2015
* Système d’exploitation : Windows 7

## L’Objectif du Projet

**Objectif principal demandé par l’école :**  Le but de ce projet est de créer un Logiciel de jeu le Baillis qui compose une interface graphique.

**Description du projet :**

* **Présentation du jeu**

La règle du jeu :

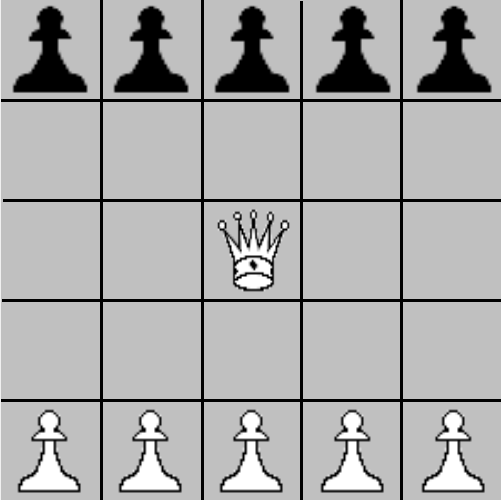


Figure  :position initiale

* Les pièces : il y a 2 types de pièces sur un damier de 5x5
* Type1 : Les pions : chaque pion a le droit de déplacer horizontalement, verticalement ou diagonalement. Chaque équipe possède 5 pions de différents couleurs soit le Noir ou le Blanc
* Type 2 : Le roi : La pièce unique du damier et il a même droit de déplacer que les pions.
* Déroulement :
* A chaque tour ; le joueur doit déplacer le roi d’abord puis un pion de son côté après. Les pions servent à empêcher les déplacements du roi. Le jeu sera terminé quand un joueur arrive à amener le Roi à son coté, donc il sera gagné. Si le roi n’a plus de possibilité pour se déplacer, le match sera nul dans ce cas.

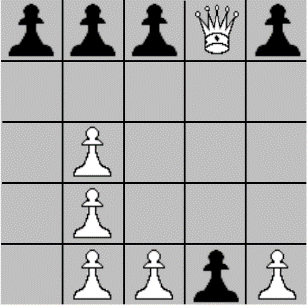
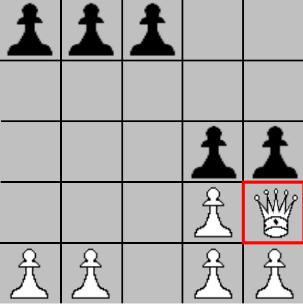
 

Figure Position gagnant d'équipe noire Figure Match nul

* **Objectif détaillés :**
* Créer une interface graphique qui contient plusieurs modes de jeux notamment, le Solo Mode, le mode en réseaux, le mode entre 2 joueurs.
* Faire le graphique, et la gestion des évènements
* Le programme doit contrôler la règle du jeu dans le cas du mode 2 joueurs
* Créer une intelligence artificielle qui permet de jouer contre l’Homme
* Gestion de la partie en Réseaux
* Un guide qui aide utilisateur pour l’utilisation la règle du jeu ainsi que l’utilisation de l’interface graphique.

## 1.3 Calendrier prévisionnel et le calendrier réel

Pour faire cela, nous réalisons un planning avec différentes étapes de développement envisagée :

Pendant le projet, il apparaît toujours les bugs ; des erreurs, par conséquence, nous prenons toujours tous les après-midis pour corriger et tester le programme au fur à mesure, cela évite de faire un grand travail qui ne marchera pas. Nous testons les modules, et après avoir validé le module, nous passons à l’étape suivante. L’intelligence artificielle est une notion nouvelle, et nous devons faire des recherches et des algorithmes qui permet de faire cela. Cette étape est la plus difficile car nous devons négliger et forcer l’ordinateur de faire un déplacement qui évite les dangers. Puis nous devrons faire une interface qui est facile à l’utilisateur à l’utiliser.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Table de répartition des tâches prévisionnelles | | | | | | | | |  |  |  |  |  |  |
| Élément, fonction | Nom de fonction | Description | 22-mai | 23-mai | 24-mai | 25-mai | 26-mai | 27-mai | 28-mai | 29-mai | 30-mai | 31-mai | 01-juin | 02-juin |
| Graphique |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | Gestion d’affichage, initialisation de plateau |  | Retard |  |  |  |  |  |  |  |  |  |  |  |
|  | Gestion des évènements, contrôle du souris |  | Retard |  |  |  |  |  |  |  |  |  |  |  |
|  | Déplacer les objets, enregistrement des pièces dans une carte |  | Retard |  |  |  |  |  |  |  |  |  |  |  |
|  | Entourer les mouvements possibles. |  | Retard |  |  |  |  |  |  |  |  |  |  |  |
|  | Vérification la fin du jeu |  | Retard |  |  |  |  |  |  |  |  |  |  |  |
|  | Changer le tour |  | Retard |  |  |  |  |  |  |  |  |  |  |  |
|  | Ai mouvement |  | Retard |  |  |  |  |  |  |  |  |  |  |  |
| Evaluation |  |  | Retard |  |  |  |  |  |  |  |  |  |  |  |
|  | Fonction évaluation |  | Retard |  |  |  |  |  |  |  |  |  |  |  |
|  | Fonction décisive de AI |  | Retard |  |  |  |  |  |  |  |  |  |  |  |
| Connection en réseaux |  | Jouer à plusieurs personnes | Retard |  |  |  |  |  |  |  |  |  |  |  |

Légende :

Etape liée à l’intelligent Artificielle

Etape liée à la graphique et la gestion des mouvements des pièces

Étape annulée dans le cas de manque de temps

Test le programme

Table de répartition de tâche réelle

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| element, fonction | Nom de fonction | 22-mai | 23-mai | | 24-mai | | 25-mai | | 26-mai | | 27-mai | 28-mai | 29-mai | | 30-mai | | 31-mai | 01-juin | 02-juin |
| Graphique |  | retard | M | A | M | A | M | A |  |  |  |  | M | A | M | A |  |  |  |
|  |  |  |  |  |  |  |  |  |  | |  |  |  | |  | |  |  |  |
|  | Gestion d’affichage, initialisation de plateau | retard |  |  |  |  |  |  |  | |  |  |  | |  | |  |  |  |
|  | Gestion des évènements, contrôle du souris | retard |  |  |  |  |  |  |  | |  |  |  | |  | |  |  |  |
|  | Déplacer les objets, enregistrement des pièces dans une carte | retard |  |  |  |  |  |  |  | |  |  |  | |  | |  |  |  |
|  | Entourer les mouvements possibles. | retard |  |  |  |  |  |  |  | |  |  |  | |  | |  |  |  |
|  | Vérification la fin du jeu | retard |  |  |  |  |  |  |  | |  |  |  | |  | |  |  |  |
|  | Changer le tour | retard |  |  |  |  |  |  |  | |  |  |  | |  | |  |  |  |
|  | Ai mouvement | retard |  |  |  |  |  |  |  | |  |  |  | |  |  |  |  |  |
| Evaluation |  | retard |  |  |  |  |  |  |  | |  |  |  | |  | |  |  |  |
|  | Fonction évaluation | Retard |  |  |  |  |  |  |  | |  |  |  | |  |  |  |  |  |
|  | Fonction décisive de AI | retard |  |  |  |  |  |  |  | |  |  |  | |  |  |  |  |  |
|  | Décoration et commenter les codes, rédiger le rapport, écrire le guide |  |  |  |  |  |  |  |  | |  |  |  | |  | |  |  |  |

## 1.4 Le choix : La description structure POO du programme.

Pour assurer la cohérence du programme, nous devons construire un mécanisme de jeux en schéma POO avec les différentes classes et le lien entre elles.

Schéma : le schéma est donné à titre indicatif, le rôle de ce schéma est pour présenter une liaison entre les classes. (Page suivante)

Dans ce schéma, nous voyons que la classe Winform CMainmenu est pour la gestion de différents modes.

Puis la classe CGraphique est parent de CBoard qui est un UserControl, toutes les interactions comme la gestion des déplacements, vérifications de la fin du programme se fassent sur la classe CBoard. La classe CGraphique est pour afficher les informations qui guident l’utilisateur.

Puis la classe CCartographie est pour stocker toutes les positions des différentes Pièces, cette classe retourne des informations et aussi affecte la nouvelle position des Pièces.

La classe CPiece contient des informations d’une Pièces.

Concernant les différentes variables, nous décidons d’affecter à chaque classe son propre variable privée nécessaire au constructeur. Surtout la classe CAnalyse, qui contient le cœur de l’IA, toutes les paramètres du jeu concernant devrait être déclarées dans la classe, même si les variables sont identiques dans la classe. Cela évite le passage par des propriétés qui rend le code très difficile à comprendre.

En fin, toutes les fonctions sont construites dans les classes à partir de la fonctionnalité de cette classe, par exemple, la classe CBoard qui gère les mouvements des pièces doit contient les fonctions qui servira à dessiner les objets, et interagir avec la classe CCartographie pour mettre à jour les positions. Toutes les fonctions seront détaillées dans la suite du rapport.

Bien sûr que les variables sont à ajouter en dehors de la prévision, de même pour les fonctions. Pourtant, il est très important de lister et de prévoir les interactions, les héritages pour suivre le bon chemin.

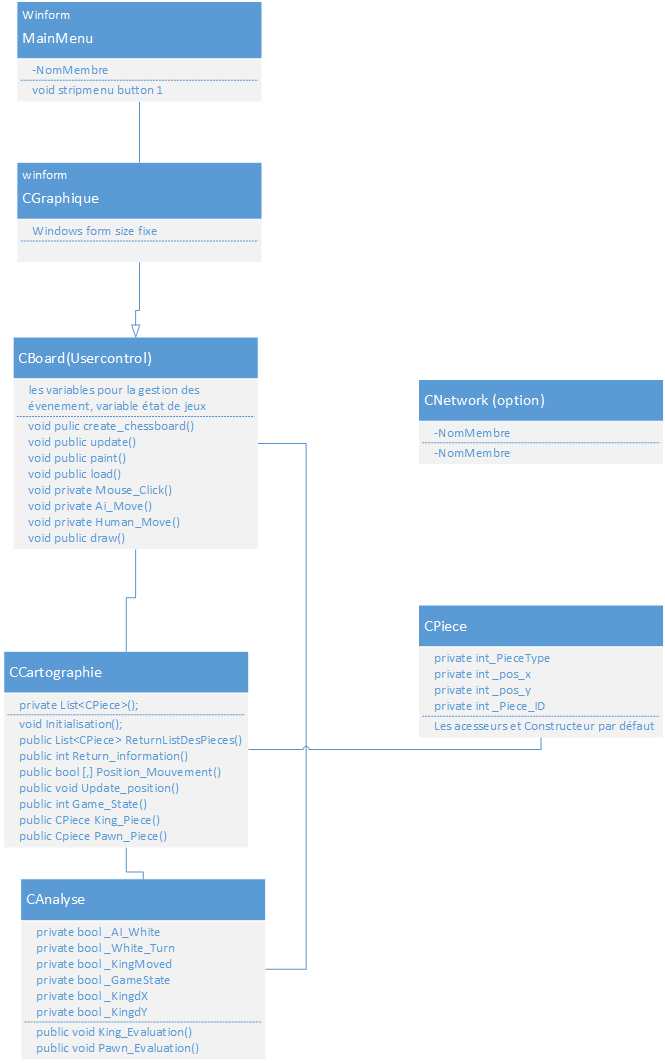


Figure Structure Général

**Structure Détaillés de chaque classe**

|  |  |
| --- | --- |
| CMainmenu | |
| Variable | |
| private bool \_Ai\_White; | Stocker paramètre du jeu |
| private string \_Name\_Player\_White | Stocker le nom de joueur |
| private string \_Name\_Player\_Black; | Stocker le nom de joueur |
| Fonction et Constructeur | |
| public CMainmenu() | Afficher le message de Guide |
| public bool Ai\_White | Assesseur |
| private void button1\_Click(object sender, EventArgs e) | Gestion de bouton, déclencher le mode Solo |
| public bool Get\_Choice() | paramètre du jeu, true si utilisateur choisit le coté noire, false sinon |
| private void button2\_Click(object sender, EventArgs e) | Gestion de bouton, déclencher le mode 2 joueurs |

Interface Graphique :class CMainmenu()

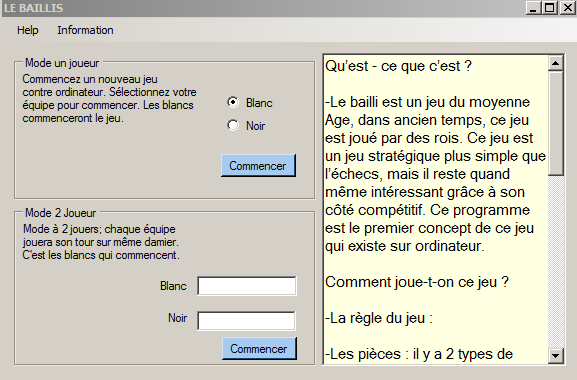


Figure Le menu

|  |  |
| --- | --- |
| CGraphique2 Parent | |
| Variable | |
| Fonction et Constructeur | |
| public CGraphique2(string Name\_Player\_1, string Name\_Player\_2) | Constructeur par paramètre (initialiser les nom des joueurs) |
| public void majlabel(string txt) | Mise à jour le label |
| private void Exit\_button\_Click\_1(object sender, EventArgs e) | Fermeture de la form |

|  |  |
| --- | --- |
| CBoard2 Enfant | |
| Variable | |
| private int W = 500;  private int H = 500;  private int Case\_x = 100;  private int Case\_y = 100;  private CCartographie m = new CCartographie();  private Point \_Mouse\_Position  private Point \_Mouse\_Position\_Select ;  private Point \_Mouse\_Position\_Depart;  private Pen \_hightlight\_pen\_Blue;  private Pen \_hightlight\_pen\_Red;  private Pen \_hightlight\_pen\_Green;  private bool \_highlight;  private bool \_WhiteTurn;  private bool \_KingMoved;  private bool \_GameState; | Dimension de Damier  Dimension de Damier  Dimension de chaque case  Dimension de chaque case  Instancier un objet de type Ccartographie  Case sélectionné  La case d'arrivée de la pièce  La case de départ pour la pièce, ou bien l'identification de la pièce  La couleur utilisée pour entourer les pièces  verifie si on a bien cliquer sur une pièces pour l'entourer.  verifie c'est le tour des blancs, sinon faux  Verifie si on a bien bougé le roi  Etat de Jeu |
| Fonction et Constructeur | |
| private void CBoard2\_Load(object sender, EventArgs e) | Initialiser les composants au démarrage |
| private void CBoard2\_Paint(object sender, PaintEventArgs e) | Initialiser interface graphique au démarrage |
| private void draw() | Dessiner les Pièces, les objets |
| private void CBoard2\_MouseClick(object sender, MouseEventArgs e) | Gestion des clique du souris, ainsi que contrôler le déroulement du jeu, la règle du jeu. |
| private void move(int Piece\_Type) | Vérifier les déplacement d’utilisateur, valider ses mouvements et mettre à jours dans la base des données |

Interface Graphique : CBoard2+CGraphique

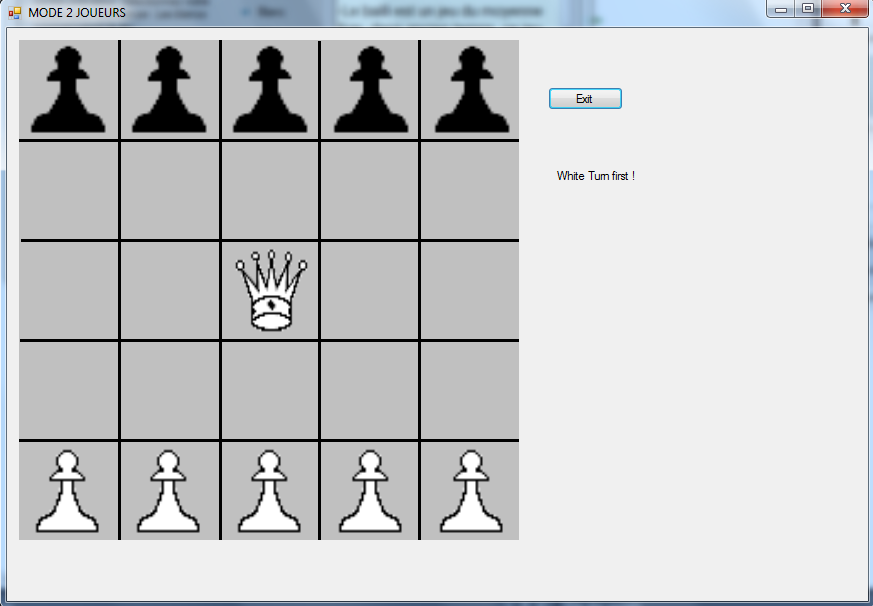


Figure Espace de jeux

MessageBox :

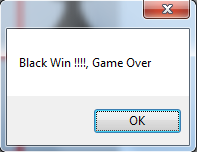


Figure Message gagnant

|  |  |
| --- | --- |
| CCartographie | |
| Variable | |
| private List<CPiece> ListPieces | La Liste contient toutes les pièces |
| Fonction et Constructeur | |
| public void Initilisation() | Initialiser toutes les pièces sur le damier |
| public List<CPiece> ReturnListDesPieces | Retourner la liste des pièces |
| public int Return\_information(int x, int y) | Retourner l’information de la pièce choisie sur le damier |
| public bool[,] Position\_Mouvement(int Piece\_Type\_Select, int x, int y) | Créer un table des mouvements possibles pour chaque pièces |
| public void Update\_position(int PieceType, int x, int y, int dx, int dy) | Mise à jour la position des pièces |
| public int Game\_state() | Retourner l’état du jeux et déterminer qui est le gagnant |
| public CPiece King\_Piece() | Retourner l’information de la pièce Roi |
| public CPiece Pawn\_Piece(int PieceID) | Retourner l’information des pièces pions |

|  |  |
| --- | --- |
| CPiece | |
| Variable | |
| private int \_PieceType  private int \_pos\_x  private int \_pos\_y  private int \_Piece\_ID | Identification différents types de pièces 1 (Roi), 2 (Pions Noirs), 3 (Pions Blanc)  Position en colonnes  Position en rang  Identification des pièces générales |
| Fonction et Constructeur | |
| public CPiece(int PieceType, int x, int y, int Piece\_ID) | Constructeur par paramètre |
| public CPiece() | Constructeur par défaut |
| public int Piece\_ID | Assesseur |
| public int PieceType | Assesseur |
| public int x | Assesseur |
| public int y | Assesseur |

Pour la partie IA : il y a des modifications mais ces classes restent la même structure que les classes pour 2 joueurs

|  |  |
| --- | --- |
| CGraphiqueSolo Parent | |
| Variable | |
| Fonction et Constructeur | |
| public CGraphiqueSolo(bool AI\_White) | Constructeur par paramètre |
| public void majlabel(string txt) | Information |
| public void debug\_label1(string txt) | afficher le mouvement du Roi fait par l'ordinateur |
| public void debug\_label2(string txt) | afficher le mouvement du Pion fait par l'ordinateur |
| public void Game\_Setting(bool AI\_White) | Configuration de AI |
| private void button1\_Click(object sender, EventArgs e) | Quitter le jeu |

|  |  |
| --- | --- |
| CBoardSolo Enfant | |
| Variables | |
| private int W = 500;  private int H = 500;  private int Case\_x = 100;  private int Case\_y = 100;  private CCartographie m = new CCartographie();  private Point \_Mouse\_Position;  private Point \_Mouse\_Position\_Select;  private Point \_Mouse\_Position\_Depart;  private Pen \_hightlight\_pen\_Blue;  private Pen \_hightlight\_pen\_Red;  private Pen\_hightlight\_pen\_Green;  private bool \_highlight;  private bool \_WhiteTurn;  private bool \_GameState;  private bool \_KingWhiteMoved ;  private bool \_KingBlackMoved;  private bool \_AI\_White; | Vérifie si le roi blanc a bougé  Vérifie si le roi noir a bougé  Vérifie si l'IA est noir ou blanc dépend de l'utilisateur |
| Fonctions et Constructeurs | |
| private void CBoard2\_Load(object sender, EventArgs e) | Initialiser les composants au démarrage |
| private void CBoard2\_Paint(object sender, PaintEventArgs e) | Initialiser interface graphique au démarrage |
| private void draw() | Dessiner les Pièces, les objets |
| private void CBoard2\_MouseClick(object sender, MouseEventArgs e) | Gestion des clique du souris, ainsi que contrôler le déroulement du jeu, la règle du jeu. |
| private void HumanMove(int Piece\_Type) | Vérifier les déplacement d’utilisateur, valider ses mouvements et mettre à jours dans la base des données |
| private void AI\_Move() | Déplacer les pièces par l’ordinateur |

|  |  |
| --- | --- |
| CAnalyse | |
| Variables | |
| private bool \_AI\_White  private bool \_WhiteTurn  private bool \_KingMoved  private bool \_GameState  private int \_KingdX  private int \_KingdY | Déplacement du roi  Déplacement du roi |
| Fonctions et Constructeurs | |
| public CAnalyse(bool WhiteTurn, bool AI\_White = false, bool KingMoved = false, bool GameState = true) | Contructeur par parametere |
| public bool White\_Turn | Assesseur |
| public bool KingMoved | Assesseur |
| public bool WhiteTurn | Assesseur |
| public bool GameState | Assesseur |
| public int Kingdx | Assesseur |
| public int Kingdy | Assesseur |
| public void King\_Evaluation(CCartographie cartographie) | Fonction qui retourne la décision de l’IA sur le roi |
| public CPiece Pawn\_Evaluation(CCartographie cartographie) | Fonction qui retourne la décision de l’IA sur les différentes pièces |

# Réalisation du projet.

Après avoir planifié et avoir eu un schéma en tête, nous décidons de développer tout d’abord l’interface graphique du jeu, puis construire la règle qui contrôle le jeu. Puis nous développons le mode à 2 joueurs pour vérifier que toutes les règles fonctionnent normalement. En fin, nous développons l’IA qui suit cette règle et jouer contre l’Homme.

## 2.1 Gestion de l’interface graphique et structure des données

### Stockage les positions des pièces, initialise le graphique, les premiers pas de construction

Pour conceptualiser cette structure, nous avons étudié le cas d’un jeu très populaire, c’est les échecs. D’après avoir étudier, nous décidons de crée la classe CPièce qui contient les 4 valeurs membres pour identifier les différents types de pièces dans le jeu.

Les 4 variables sont :

CPièce.cs

private int \_PieceType; // si 1 (Roi), 2 (Pions Blancs), 3 (Pions Noirs)

private int \_pos\_x; // position en x des pièces

private int \_pos\_y ; //position en y des pièces

private int \_Piece\_ID; // Identifiants des pièces (0-4) Pions Blancs, (5-9) pions noirs, 10 Roi

Puis nous mettons les constructeurs par défaut, les constructeurs avec les paramètres, et les propriétés pour affecter les valeurs dans cette classe.

Ensuite, nous créons la classe CCartographie qui gère la <List> des objets CPiece.

CCartographie.cs

private List<CPiece> ListPieces;

public CCartographie()

{

ListPieces = new List<CPiece>();

}

Puis nous allons initialiser cette liste avec les différentes Pièces sur le damier avec les différentes coordonnées.

CCartographie.cs

public void Initilisation() //initialisation des Pièces dans le damier

{

ListPieces.Add(new CPiece(2, 0, 0, 0)); //pion noir

ListPieces.Add(new CPiece(2, 1, 0, 1)); //pion noir

ListPieces.Add(new CPiece(2, 2, 0, 2)); //pion noir

ListPieces.Add(new CPiece(2, 3, 0, 3)); //pion noir

ListPieces.Add(new CPiece(2, 4, 0, 4)); //pion noir

ListPieces.Add(new CPiece(3, 0, 4, 5)); //pion blanc

ListPieces.Add(new CPiece(3, 1, 4, 6)); //pion blanc

ListPieces.Add(new CPiece(3, 2, 4, 7)); //pion blanc

ListPieces.Add(new CPiece(3, 3, 4, 8)); //pion blanc

ListPieces.Add(new CPiece(3, 4, 4, 9)); //pion blanc

ListPieces.Add(new CPiece(1, 2, 2, 10)); //pion blanc

}

### Initialiser le damier et l’interface graphiques.

**Création de Menu**

Après avoir créé un espace de gestion nous allons procéder la partie graphique, en effet, nous allons créer une Winform CMainmenu qui contient les commandes des types des jeux et le guide pour le jeu. Pour faire cela, nous utilisons les outils de Winform.

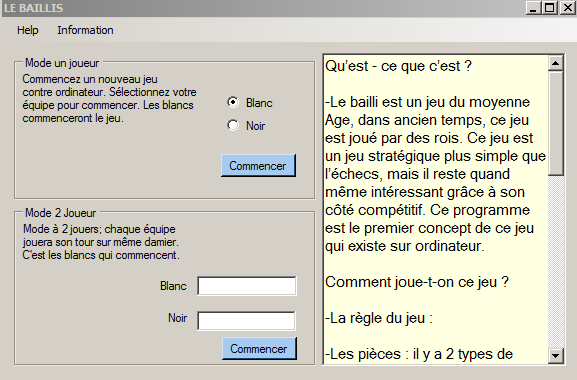


Figure Menu Principal du Programme

Puis nous créons 2 Usercontrols : CBoardSolo (Pour IA) et CBoard2

Puis 2 winforms diférents qui contiennent les 2 types Usercontrols différents : CGraphique2.cs, CGraphiqueSolo.cs (Pour IA).

**Création d’interface Graphique et initialisation du damier**

Dans le Usercontrol CBoard2, nous allons écrire le code pour affichage de damier et aussi les pièces.

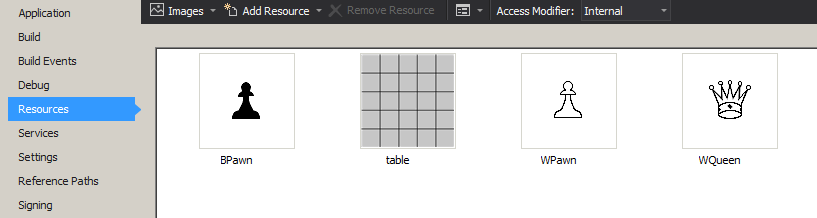
Pour avoir le damier, nous avons 2 solutions, soit nous créons par les boucles et les rectangles, soit nous dessinons dans Paint un damier de 500px ;500px. Nous choisissons la solution de dessiner dans Paint un damier. Puis nous aurons besoins ainsi les images des pièces, pour cela, nous prenons les images sur internet. Nous ajoutons dans le dossier ressource les images comme suivant : 

Figure Ressource des pièces

Puis nous initialisons les composants, en créant une fonction de dessin qui s’appelle draw() pour mise à jour tous les temps les pièces et le damier. Nous plaçons cette fonction dans l’évènement Load() du Usercontrol CBoard.

Pour effectuer ce code nous devrons écrire une fonction qui permet de retourner les types de pièces à différents endroits de damier dans la classe CCartographie

CCartographie

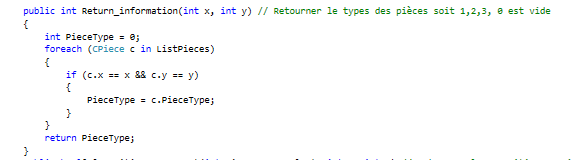


Figure Retourner la PieceType

CBoard2



Figure Code pour dessiner et placer les pièces

En résultat, nous obtenons comme ceci

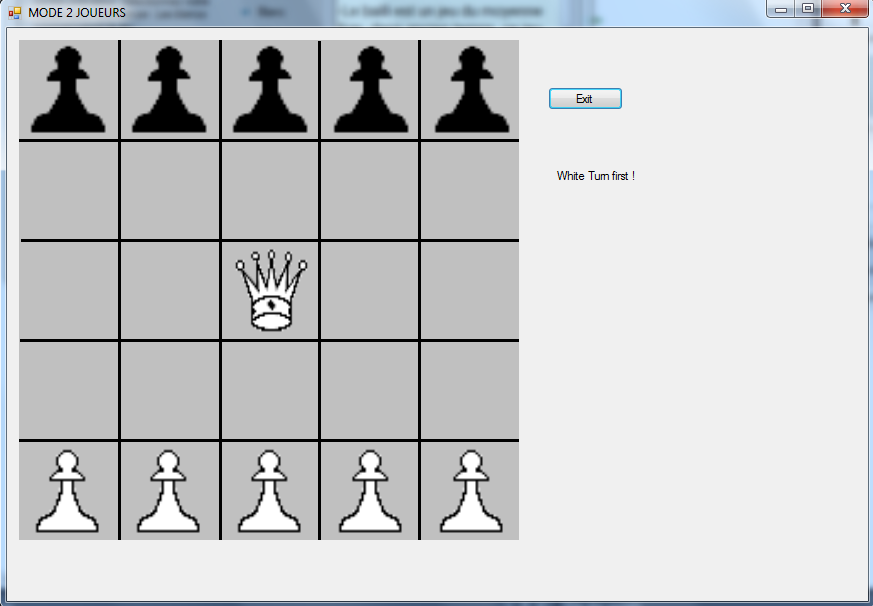


Figure Interface Graphique principale

### Mis à jour les mouvements des pièces, appliquer la règle du jeu dans le programme, création de la partie 2 joueurs

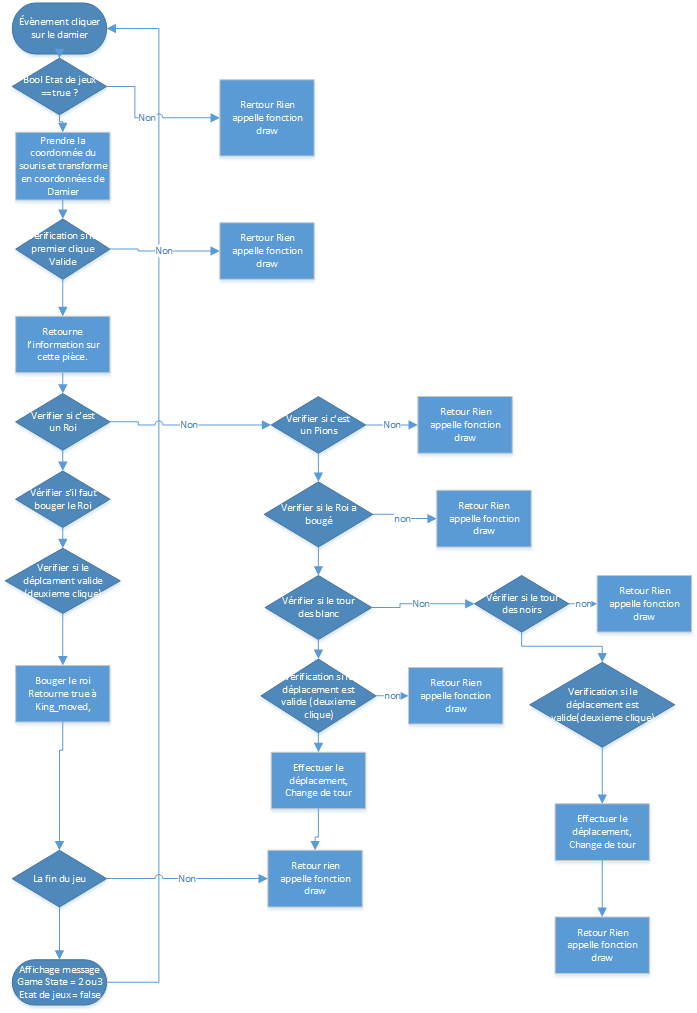
****

Figure Schéma mécanisme du jeux, gestion graphique

Pour construire ce mécanisme, nous utilisons les fonctions suivantes :

Classe CBoard2

private void CBoard2\_MouseClick(object sender, MouseEventArgs e) C’est la gestion de l’évènement « cliquer sur le damier » dans le schéma.

Puis dans la classe CCartographie, nous construisons les fonctions suivantes :

public int Return\_information(int x, int y) retourner le type de pièce.

public bool[,] Position\_Mouvement(int Piece\_Type\_Select, int x, int y) retourner une table des mouvements possible liée à la pièce.

public void Update\_position(int PieceType, int x, int y, int dx, int dy) Mise à jour de nouvelle position de la pièce.

public int Game\_state() Retourner l’état du jeu.

Qui gère tous les traitements d’information de cette classe, puis pour retourner les différentes informations vers la classe principale.

Dans la classe principale CBoard2, nous créons la fonction public void Move() qui appelle les méthodes de la classe CCartographie que nous avons instancié au début de la classe CBoard2. Cette fonction void Move() vérifie si le mouvement fait par l’utilisateur est valide et elle gère aussi le tour de chaque équipe.

Nous ajoutons cette fonction dans la fonction private void CBoard2\_MouseClick(object sender, MouseEventArgs e), l’appelle à l’endroit précis, notamment dans la condition du 2e clique pour sélectionner destination de la pièce.

## Créations de L’IA

### Transforme la structure d’affichage pour l’IA.

Pour avoir une structure qui inclut un Intelligent Artificiel (IA), nous devrons changer la structure des données qui permet de la compréhension de l’IA du problème.

La classe CBoardSolo est presque identique que la classe CBoard2, mais on distingue le coup de l’homme et l’IA en 2 fonctions élémentaires différentes : HumanMove() qui gère les coups de l’Homme et AI\_Move() qui gère les coups de l’IA.

Nous ajoutons dans la Classe 3 nouvelle variables :

private bool \_KingWhiteMoved; // Vérifie si le roi noir a bougé

private bool \_KingBlackMoved; // Vérifie si le roi noir a bougé

private bool \_AI\_White; // le côté que l’IA va jouer

Puis nous modifions dans la partie gestion des évènements du souris 2 cases différentes qui sont dépendantes de \_AI\_White.

Puis après le coup de l’utilisateur, nous mettons l’appelle à la fonction AI\_Move() pour qu’il effectue les mouvements.

Dans la fonction Ai\_Move, nous créons les conditions de jeu comme si l’IA est un joueur adversaire de l’Homme. C’est-à-dire, si le joueur choisis les blancs, la condition pour l’équipe noire doit être appliqué sur l’IA grâce à des conditions.

### Réalisations des algorithmes aléatoires contrôlées de l’ordinateur

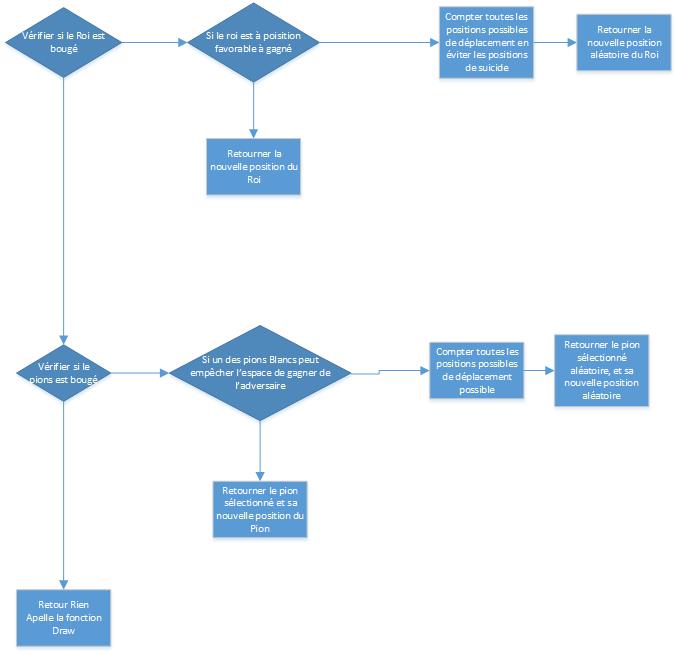


Figure Schéma logique de l'IA

Pour effectuer cela, dans la classe CAnalyse, nous écrivons les 2 fonctions King\_Evaluation et Pawn\_Evaluation qui retournent les coups joués.

Cette classe sera instanciée dans la fonction AI\_Move avec les paramètres du jeu telles que \_WhiteTurn, \_AI\_White, \_KingBlackMoved, \_GameState. Pour établir une stratégie adaptée à la situation. Puis nous appellons les fonctions dans l’ordre, en effet, Nous appelons la fonction King \_Evaluation, d’abord puis la fonction Pawn\_Evaluation par la suite. Les valeurs retours de ses 2 fonctions sera mis dans la fonction pour la mise à jour de la position (Update\_Position ()). Puis nous appelons la fonction draw() pour la mise à jour de l’interface graphique.

# Déroulement du projet

## 3.1. Le travail réalisé

Dans le premier jour, nous avons établi un planning prévisionnel pour les différentes étapes de développement avec des objectifs différents. Puis le deuxième jour, nous commençons le développement. Pour chaque nouvelle méthode écrite, il faut penser aux variables à déclarer, puis les organiser tel que c’est le plus facile à lire et corriger. Chaque fonctionnement est lié à une fonction différente, et nous essayons le plus d’optimiser le code, c’est-à-dire d’éviter les répétitions. Si une fonction est finie, nous demandons le conseil de l’encadrant pour qu’il nous donne les conseils d’amélioration. Le 30 Mai 2017, le programme principal est presque fini, les gros bugs sont presque corrigés et il nous reste que la partie décoration à finaliser dans la suite. Puis le 1 juin 2017, nous réalisons le rapport avec le diaporama pour le jour de la soutenance.

## 3.2 La validation de code

Pour chaque fonction, pour chaque fonction écrite, il faut la vérifier et corriger tous les bugs qui peuvent être produits pendant le développement. Le moyen que nous avons utilisé est d’utiliser le débogage de Visual Studio et regardons toutes valeurs des variables pour déterminer d’où vient cette erreur.

Une autre méthode que nous avons utilisée est de créer un label et suit tous les changements des positions des Pièces, cela me permet de deviner directement d’où vient cette erreur du programme.

Comme le programme est un jeu de stratégie, donc la liaison entre les classes et l’interface graphique est très forte. C’est à cause de cela, nous n’écrivons pas tout le programme dans une fois et la testons, mais écrivons chaque fonction et la testons à la suite pour éviter d’avoir un gros programme avec plusieurs erreurs inconnues.

Parfois, nous devons corriger les erreurs algorithmiques, pour contrôle ce problème, nous devons rétablir les conditions, et pensons quels sont les cases exceptionnelles qui provoque des erreurs puis étrécir les conditions pour que la fonction marche.

## 3.3 Les difficultés rencontrés

La grande difficulté de ce projet est la partie de l’IA, en effet, nous n’avons pas encore la notion de l’intelligent Artificielle et sa structure dans la formation. Donc nous devons rechercher les documents supplémentaires et réalisons que pour faire un IA, il faut appliquer plusieurs algorithmes différents. Mais le temps de réalisation de notre projet est court donc c’est très difficile pour construire une véritable IA. Donc nous utilisons le processus aléatoire pour faire des décisions de l’IA. Mais parfois, ces processus provoquent des Erreurs imprévisibles qui sont très difficiles à trouver la source. Pour fixer cela, nous devons appliquer plus de condition pour limiter cela.

## 3.4 Etat final du projet

Ce Jeu est totalement nouveau sur ordinateur, et ce programme est le premier concept de ce jeu sur les ordinateurs. Donc nous considérons que cela est une réussite, pourtant, il reste toujours les faiblesses et les éléments à corriger.

Après 2 semaines, nous pouvons récapituler résultat comme le suivant :

|  |  |
| --- | --- |
| Réussi | A améliorer |
| * Interface Graphique facile à l’utiliser * Ludique * Fonctionnement irréprochable * Mode de fonctionnement entre 2 joueurs * Mode contre les ordinateurs | * Créer une partie en réseaux * Affiner intelligent Artificielle avec les algorithmes MiniMax * Réduire les codes répétitifs |

## 3.5 Cahier des charges pour les éventuelles successeurs.

Ce programme permet d’envisager de nombreuses améliorations. Comme ce jeu est un nouvel modèle sur ordinateur, donc l’étude sur l’arbre de décision est nécessaire et rendre la IA plus efficace. De plus, construire une table de transposition permet de réduire le temps de calcul du programme. Le problème de ce projet est que nous n’avons pas avoir suffisamment du temps pour construire des parties en réseau. S’il y a cette fonctionnalité, le jeu sera plus intéressant et sociable.

Pour récapituler : Les tâches à développer ce programme est le suivant :

* Ecrire une classe de réseaux pour le programme qui permet de jouer entre 2 personnes à distances
* Créer plusieurs fonctions et réduire la taille de code
* Faire une étude sur la stratégie du jeu, construire des fonctions évaluations plus performants pour avoir une bonne décision de l’IA.
* Construire un algorithme de Minimax pour améliorer ce programme.

# Conclusion

Ce projet était pour moi l’occasion de progresser dans plusieurs domaines, c’est une expérience très enrichissante. Avec le développement de la technologie, et la science informatique, ce programme présente pour nous comme une démonstration de ce que l’IA peut faire dans la vie humaine. De plus, nous arrivons à développer, conceptualiser et modélisation un jeu du Moyen Age sur l’ordinateur est un résultat très grand que nous n’avons jamais fait auparavant. Pour réaliser ce logiciel, nous devons respecter les étapes de travail avec le temps précis, pour ne pas tromper sur la direction. Finalement, nous arrivons à finir le programme comme marquer dans le planning et nous sommes satisfaisons du résultat obtenu.

Cependant, le temps du projet est assez court, nous n’avons pas le temps de développer la partie en réseau de ce jeu. Le programme reste encore plusieurs améliorations à faire pour assurer une bonne fonctionnalité, mais nous pensons que nous allons développer, ajouter encore des fonctionnalités pour perfectionner le programme après et le mettre en ligne.