

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de la Manouba
École Nationale des Sciences de l'Informatique



Rapport de Projet du module de Programmation

SUJET :

Jeu d'Inversion
(Lights Out)

Réalisé par :

Wicem Zrelly

Mohamed Said Mezghanni

Jihed Sadaoui

et Wissem Gouja

Sous l'encadrement de :

M. Mejdi Jribi

Année universitaire :

2011-2012

Table des matières

Introduction générale	1
1 Historique et règles de jeu d'inversion "Lights out"	2
1.1 Histoire de Lights out	2
1.2 Règle du jeu	4
2 Analyse et Spécifications des besoins	5
2.1 Besoins fonctionnels	5
2.2 Besoins non fonctionnels	6
3 Conception	8
3.1 Décomposition du problème en sous-modules	8
3.1.1 Etude théorique de la solution	8
3.1.2 L'approche Orientée Objet	9
3.1.3 Description des sous-modules	9
3.2 Diagramme de modules	9
3.3 Les structures de données utilisées	10
3.3.1 Structure de données de la partie abstraite	11
3.3.2 Structure de donné de la partie graphique	11
3.4 Algorithme de la solution	11
3.4.1 Algorithme du programme principale	11
3.4.2 Procédure solvable	12
3.4.3 Procedure solve	13
4 Réalisation	15
4.1 Environnement de travail	15

TABLE DES MATIÈRES

4.1.1	Environnement de travail matériel	15
4.1.2	Environnement de travail logiciel	15
4.2	Justification du choix technologiques	16
4.3	Aperçu sur le travail réalisé	16
Conclusion Générale		23

Table des figures

1.1	Lights Out Classic	3
1.2	Lights Out Cube	4
3.1	Diagramme de modules	10
4.1	Ecran d'accueil	16
4.2	Interface d'accueil	17
4.3	Le bouton Mix	18
4.4	Le bouton Edit	18
4.5	Alerte : grille insoluble	19
4.6	Le bouton Reset	19
4.7	Le menu de choix de grille	20
4.8	Le bouton Play	20
4.9	Le bouton Solve	21
4.10	Fenêtre Win	21
4.11	Fenêtre Lose	22
4.12	Le menu Help	22
4.13	Le menu Quit	23

Introduction Générale

Les jeux vidéo attirent désormais un public de plus en plus large et séduisent également de nombreux développeurs. Malheureusement, la programmation de jeu vidéo est souvent méconnue et beaucoup imaginent qu'elle est aussi facile que d'y jouer. En effet, le jeu vidéo demande beaucoup d'investissement ainsi que des connaissances théoriques et pratiques assez poussées, que la plupart des étudiants n'ont forcément pas. L'évolution des jeux vidéo est le fruit de plusieurs années de travail dans le domaine de développement des logiciels. Bien que la plupart des premiers jeux vidéo étaient assez simplistes et d'une jouabilité presque nulle, plusieurs ont suivi le fil de l'évolution exponentielle des technologies informatiques et sont devenus cultes. Ces derniers ne cessent d'être hérités de génération en génération. Désormais, chaque jeu ayant un succès a son équivalent sur les ordinateurs. On en cite principalement "Monopoly", "Lights out ". C'est dans ce cadre que s'inscrit la réalisation de notre projet qui consiste à l'implantation du jeu d'inversion dans une version avancée par rapport à l'existence. Le présent rapport est organisé comme suit : nous présentons dans le premier chapitre un historique du jeu ainsi que ses règles. Un second chapitre, sera consacré à l'analyse et la spécification des besoins de l'application. Dans le troisième chapitre, nous détaillerons la partie conception du projet. Le quatrième chapitre rend compte de la réalisation de l'application par la mise en oeuvre d'interface de jeu.

Chapitre 1

Historique et règles de jeu d'inversion "Lights out"

Introduction

Le présent chapitre est organisé en deux parties : Dans une première partie nous présentons un bref historique du jeu d'inversion. Une seconde tient compte des règles de jeu.

1.1 Histoire de Lights out

Tout a commencé en 1989, quand Avi Olti, un ingénieur en logiciel d'origine Hongroise, faisait une promenade nocturne avec chien. En marchant, Avi s'est retrouvé à essayer de résoudre un bug qu'il avait eu dans un programme. Une idée lui est venue que ce bug pourrait être un jeu vraiment sympa. Cette nuit-là Avi a écrit les premières lignes de l'algorithme de Lights Out. Zvi Herman, un ingénieur en électronique qui a travaillé avec Avi dans la même entreprise, le rejoigne, et une nuit, ils ont construit le premier modèle de Lights Out, sur la base de microprocesseur Intel 8051. SQUARICA était le nom qu'ils ont donné à leur nouveau-né et une entreprise leur proposa d'adopter leur "bébé". Malheureusement les négociations ont échoué, et le projet est mis à côté.

En Octobre 1991, Avi rencontre le Dr Gyora Benedek, un informaticien travaillant dans la même compagnie que lui. Avi a montré à Gyora le jeu. Impressionné, Gyora adopte le projet et le relance. Il a complété une solution mathématique et a trouvé un algorithme pour résoudre n'importe quel puzzle. Cet algorithme intelligent a permis l'ajout de plusieurs autres fonctionnalités

pour le jeu tel que l'aide dans la résolution et le nouveaux modes de jeu.

En 1994, un peu par hasard, Avi Weiner et Michael Ganor présente "SQUARICA" à une société de jouets. Mickey, un designer industriel, a apporté quelques nouveaux concepts de mécanique et de design, qui ont donné au projet une nouvelle âme. Avi, restant proche de son projet, développait une interface simple et Zvi a écrit un manuel de l'utilisateur. En 1995, lorsque le Salon du jouet à Nuremberg, Allemagne, SQUARICA a vécu un succès incomparable. Randy Rissman, le propriétaire de la société TIGER Toys, décide d'adopter le jeu. Stewart Sims, Tiger Toys VP Marketing, le soutenu en lui disant : "J'avais cherché un puzzle électronique pendant des années, et enfin j'ai trouvé quelque chose ingénieux" Son partenaire Roger Schiffman de Randy, a donné un appui précieux aussi. Ainsi Tiger Toys a amené au monde le jeu magnifique, Lights Out.

Des mathématiciens et les assistants ont réalisé tant de recherches sur ce jeu. À l'Université du Wyoming, le professeur Leslie Shader utilise ce jeu pour illustrer les principes de la théorie des matrices.

Lights Out a inauguré une nouvelle catégorie de jeu "Serious Fun" créant une combinaison incroyable entre les mathématiques et le plaisir.



FIGURE 1.1 – Lights Out Classic

1.2 Règle du jeu

Lights out est un jeu de réflexion. Ce jeu est constitué d'une matrice, typiquement 5x5, qui contient des cases de deux couleurs (ou des lampes avec deux état éteint et allumé). Le but du jeu est simple, il consiste à rendre toutes les cases de la grille de même couleur en cliquant case par case. Si on clique sur une case en jouant, cette case ainsi que les quatre cases adjacentes changent de couleur inversement. Cette opération est répétée jusqu'à atteindre l'objectif du jeu. Dans des versions plus avancées du jeu, il y'a des exemples avec des cubes et plusieurs autres

formes géométrique. Il y'a aussi des exemples avec plus que deux couleurs.

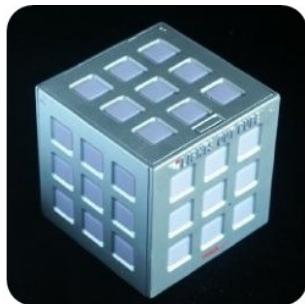


FIGURE 1.2 – Lights Out Cube

Conclusion

Lors du dernier chapitre, on a évoqué l'historique du jeu et son origine hongroise suivi des règles du jeu. Nous passons alors au chapitre suivant pour spécifier et analyser les besoins de l'application.

Chapitre 2

Analyse et Spécifications des besoins

Introduction

Ce chapitre est consacré à la spécification des besoins de l'application. Les besoins fonctionnels ainsi que non fonctionnels seront détaillés.

2.1 Besoins fonctionnels

- Le système doit permettre à l'utilisateur le lancement d'une nouvelle partie :

Un système de jeu doit permettre aux capacités individuelles de s'exprimer pleinement. Pour être efficace, ce système doit réinitialiser aléatoirement la grille lors de chaque lancement de partie. Selon son choix, l'utilisateur peut choisir une grille prédéfinie en se basant sur la difficulté.

- Le système doit contenir des cases de deux couleurs différentes :

La nécessité de passer d'une grille de début à une autre de fin impose que le jeu doit fonctionner sur le principe de deux couleurs : une couleur qui représente une lampe allumée et une autre différente de la première qui représente une lampe éteinte sachant que par un simple clic sur une case il est possible de passer cette case et celles qui les entourent d'une couleur à l'autre.

- Le système doit permettre à l'utilisateur de sélectionner une case par la souris :

Le système doit permettre d'appuyer sur une case choisie par l'utilisateur. Il suffit de sélectionner la case que doit générer le bouton lorsqu'il est pressé.

- Le système doit assurer l'inversion automatique des couleurs de la case sélectionnée par

la souris et des quatre cases qui lui sont adjacentes verticalement et horizontalement : Cette démarche doit être réalisée dans chaque sélection de case. L'utilisateur définit une grille de sélection, alors il permute de couleur avec les autres cases qui lui sont adjacentes.

- Le système doit fournir au joueur une solution optimale à la demande :

Il est bien connu que les jeux de réflexions nécessitent en général de la patience pour les terminer. C'est pourquoi notre but était de trouver la solution optimale à ce jeu d'inversion. Il fallait également minimiser le temps de recherche : une exploration "naïve" de toutes les combinaisons possibles était donc exclue. La solution sera donc basée sur une résolution mathématique qui va être traduire en un programme informatique. Dès que le joueur clique sur le bouton résolution, le système vérifie si la grille à ce moment-là admet une solution. Dans ce cas, les cases aboutissant à la résolution auront un arrière-plan lumineux différent des autres cases. Le cas échéant un message d'erreur sera affichée.

2.2 Besoins non fonctionnels

Nous présentons dans cette partie les besoins non fonctionnels relatifs à l'application.

- Le jeu doit permettre à un joueur de s'enregistrer, créer un profil et jouer des parties :

Chaque joueur aime bien jouer avec son nom afin de battre les meilleurs scores et devenir le premier dans le classement c'est pour cela qu'un formulaire ou une identification sera requise avant le lancement de chaque partie demandant les informations personnelles nécessaires au joueur.

- L'interface graphique doit être conviviale, donnant envie à l'utilisateur de découvrir le jeu :

"Ce qui est beau, c'est ce qu'on aime" Les graphismes font partie de la première chose qu'un joueur peut remarquer, un graphisme développé incite les joueurs à voir et à essayer le jeu. Bref une interface attrayante maximise l'attention et le plaisir du joueur.

- Le menu du jeu doit offrir un guide d'utilisation sous forme de menu aide :

Essayer un jeu sans le comprendre dès le début serait non seulement une perte de temps mais aussi très ennuyant. Un bouton d'aide sera alors ajouter pour informer le joueur. D'une part, sur les commandes nécessaires pour le déroulement du jeu et d'autre part, l'objectif du jeu d'une façon simplifié.

- Le menu du jeu doit permettre une prise en main aisée :

Jouer sur un ordinateur n'est pas toujours agréable, avec tous ces boutons et touches du

clavier. Plusieurs préfèrent utiliser la souris puisqu'elle est facile à manipuler : il suffit de mouvoir la souris pour déplacer le curseur sur l'écran. Dans notre cas, le jeu d'inversion, le clique sur une case engendre automatiquement le changement de couleurs des cases concernées d'où le choix de la souris.

- L'interface graphique doit offrir des produits multimédia (son ,vidéo et image) de qualité.

Conclusion

Nous avons donc éclairci les besoins fonctionnels et non fonctionnels de notre application. Nous avons également analysé ces besoins à l'aide d'un diagramme UML de cas d'utilisation. Maintenant, nous pouvons passer à l'étape suivante de ce projet qui est la conception du jeu.

Chapitre 3

Conception

Introduction

Dans cette partie nous allons commencer par décrire la solution théorique et expliciter l'approche adoptée pour développer ce jeu. Ainsi nous détaillerons la décomposition de notre application en sous-module, à travers le diagramme de module et nous finirons par spécifier les structures de données et les algorithmes utilisés.

3.1 Décomposition du problème en sous-modules

Avant tout travail de développement, il faut commencer par effectuer une étude théorique pour aboutir à une conception parfaite du problème posé.

3.1.1 Etude théorique de la solution

La première question qu'on doit se poser lorsqu'on entame le développement du jeu Inversion est : " De quoi se compose ce jeu ? " Nous avons alors décomposé l'application en deux grandes parties :

- **La partie graphique** : C'est celle qui va générer l'interface du jeu. Cette partie est modélisée par plusieurs classes et méthodes qui vont permettre l'affichage des fenêtres de dialogue et l'affichage de la fenêtre principale qui contient la grille.
- **La partie conceptuelle ou abstraite** : Elle effectue tous les calculs et les tests nécessaires à l'aide de matrices pour assurer le bon fonctionnement du jeu. Enfin, cette partie renvoie son résultat à la partie graphique pour que les composantes du jeu soient correctement

affichées à l'écran.

3.1.2 L'approche Orientée Objet

Le monde dans lequel on vit est un monde d'objet. Dans les débuts du développement informatique, modéliser les détails de la vie quotidienne était un fardeau d'importance avec la programmation modulaire, unique approche de cette époque. Néanmoins l'idée de modéliser tout en objet, en structure de donnée un peu spéciale était déjà présente, jusqu'à ce que la programmation orienté objet voit le jour. L'idée est de monter dans le niveau d'abstraction pour modéliser un objet de la vie réelle en un objet informatique, ce qui facilite en grande partie le développement surtout lorsqu'il s'agit des jeux vidéo. On se basera sur cette approche pour toute la partie graphique du jeu. Mais en ce qui concerne la partie abstraite nous allons utiliser une approche plutôt modulaire.

3.1.3 Description des sous-modules

Le jeu se décompose en plusieurs sous modules dont chacun a une fonctionnalité bien déterminée. Commençons par " génération " qui permet de générer une grille de boutons de deux couleurs différents représentant soit des lampes allumées soit des lampes éteintes.

Passons à " Solvobal " : ce sous module permet de vérifier si une grille admet une solution ou non. Un parcours de la matrice s'effectue pour vérifier cette condition. Si elle n'admet pas de solution le sous module " mix " se charge de créer une autre grille qui admet une solution selon les règles de notre jeu.

Le sous module " play " est celui qui gère la jouabilité de notre application. A chaque fois qu'on clique sur un bouton, son couleur et la couleur des boutons adjacents change vers la couleur contraire jusqu'à rendre tous les boutons dans la même couleur et là on gagne.

Le sous module " solve " permet de trouver une solution pour la grille courante en marquant les boutons ou on doit cliquer pour gagner alors que le sous module " edit " permet de générer une grille selon votre choix.

3.2 Diagramme de modules

La décomposition du jeu en sous module nous permet de dessiner le diagramme suivant :

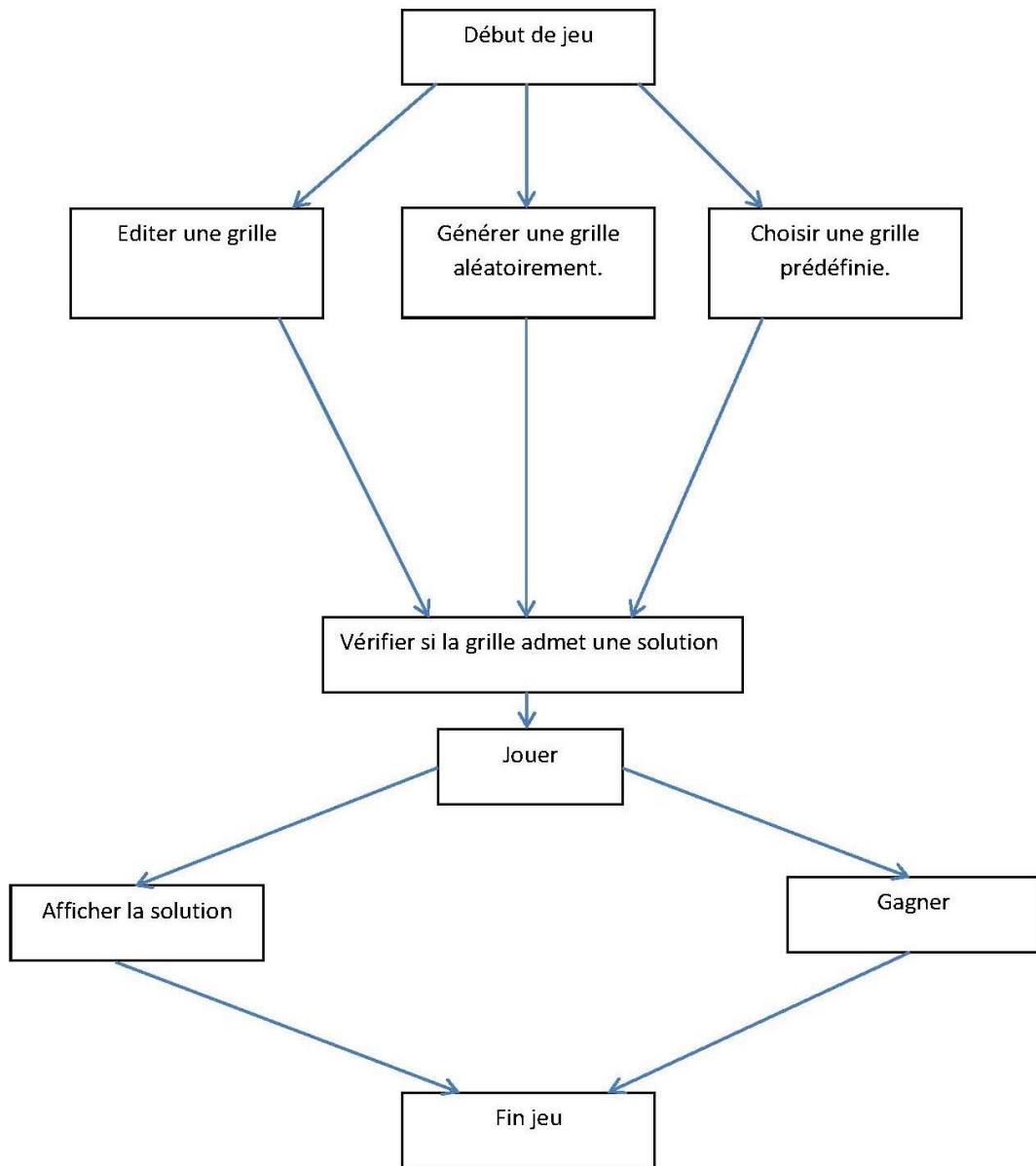


FIGURE 3.1 – Diagramme de modules

3.3 Les structures de données utilisées

Les structure de donnée utiliser peuvent se décomposer en 2 partie selon la décomposition logique de l'application.

3.3.1 Structure de données de la partie abstraite

- **n1 , n2** :pointeur sur vecteur d'entier de dimension 25
- **grid[5] [5]** : matrice de deux dimensions des entiers
- **sol[5][5]** :matrice du deux dimensions des entiers
- **v** :pointeur sur vecteur d'entier de dimension 25
- **src , dest** : pointeur sur vecteur d'entier de dimension 25

3.3.2 Structure de donné de la partie graphique

La partie graphique est la partie orienté objet par excellence , en effet chaque fenêtre représente une classe d'où les classes suivantes :

- le menu principale
- le menu quitter
- le menu du help
- le menu du jeu

3.4 Algorithme de la solution

Dans cette partie on va s'intéresser principalement aux modules qui concernent la partie abstraite.

3.4.1 Algorithme du programme principale

```
Début Programme principale() /* 2 vecteurs qui vont server a la résolution et a la vérification si la grille est résoluble */
int n1[25] = [1,0,1,0,1,1,0,1,0,1,0,0,0,0,0,1,0,1,0,1,1,0,1,0,1];
int n2[25] = [1,1,0,1,1,0,0,0,0,1,1,0,1,1,0,0,0,0,1,1,0,1,1];
int grid[5][5];
faire [ pour(int i=0 ;i<5 ;i++)faire
pour(int j=0 ;j<5 ;j++)faire
grid[i][j]=(rand()%2);
finpour.
Finpour.
]tantque (solvable est faux);
```

Afficher la grille ;
Tant que (grille non resolu) faire
Si (le joueur demande une solution) alors
solve(sol,grid,n1,n2) ;
K=vrai ;
finsi
Si (k=vrai) alors afficher la grille sol ;
Sinon afficher la grille grid ;
Finsi.
saisir ligne ;
Saisir colonne ;
Effectuer l'inversion ;
Fintantque.
Si (grille resolu k=faux) Alors
ecrire(" felicitation vous aver gagner ")
Sinon ecrire (" vous avez perdu ")
Finsi.
Fin programme principale.

3.4.2 Procédure solvable

```
Procedure solvable(int grid[][5],int * n1,int * n2)
debut
int cur[25];
int nbr;
/*on extraie a partir de la grille un vecteur de dimension 25*/
pour(int i=0 ; i<25 ; i++) faire
si (grid[i mod 5] [i/5]) alors
cur[i] = 1 ;
Sinon cur[i] = 0 ;
finsi
finpour
nbr = 0 ;
```

```
nbr =produit scalaire (n1,cur)
si (nbr !=0) alors retourner faux ;
finsi.

nbr = 0 ;

nbr =produit scalaire (n2,cur)
si (nbr !=0) alors retourner faux ;
sinon retourner vrai ;
finsi.

fin solvable.
```

3.4.3 Procedure solve

```
Procedure solve (int grid[][5],int * n1,int * n2)
```

```
Debut solve
```

```
int hintmatrix[23][23] = /*matrice 23*23 qui va nous permettre de determiner la solution*/
int cur[23];
int hintvector[25];

/*on extraie a partir de la grille un vecteur de dimension 25*/
pour(int i=0 ; i<25 ; i++) faire
si (grid[i mod 5] [i/5]) alors
cur[i] = 1;
sinon
cur[i] = 0;
finsi
finpour

/*determination d'une solution*/
pour (int i=0 ; i<23 ; i++) faire
pour (int j=0 ; j<23 ; j++) faire
hintvector[i] =(hintvector[i] + cur[j]*hintmatrix[i][j])mod 2;
finpour.
finpour.

/*determination de la solution optimale*/
```

```
int optimal[25];
int aux[25];
/*addition et copie se font composante par composante */
copier(hintvector,optimal);
copier(hintvector,aux);
additionner(aux,n1);
si ( nombrede1(aux) < nombrde1(optimal) ) alors
copier(aux,optimal);
finsi.
additionner(hintvector,aux);
additionner(aux,n2);
si ( nombrede1(aux) < nombrde1(optimal) ) alors
copier(aux,optimal);
finsi.
copier(hintvector,aux);
additionner(aux,n1);
additionner(aux,n2);
si ( nombrede1(aux) < nombrde1(optimal) ) alors
copier(aux,optimal);
finsi.
fin solve.
```

Conclusion

Au cours de ce chapitre, nous avons traité la partie conception de notre projet. Nous avons également donné la décomposition modulaire de notre application pour arriver aux algorithmes et aux structures de données utilisés. La partie conception étant achevée, nous pouvons passer à la partie réalisation de l'application dans le chapitre suivant.

Chapitre 4

Réalisation

Introduction

Ce chapitre sera dédié à l'étude du travail réalisé. Ainsi nous allons présenter l'environnement de travail puis présenter l'application et enfin visualiser quelques résultats de l'application du jeu.

4.1 Environnement de travail

Dans cette section, nous allons présenter l'environnement de travail matériel et logiciel que nous avons utilisé pour la réalisation de notre jeu d'inversion.

4.1.1 Environnement de travail matériel

Pour mener à terme ce travail, nous avons utilisé comme environnement matériel un ordinateur Lenovo ayant les caractéristiques suivantes :

- Processeur : Intel Core 2 Duo CPU T6500 @2.10GHz.
- Mémoire : 4Go de RAM.
- Disque Dur : 320 Go.

4.1.2 Environnement de travail logiciel

Tout au long du développement de notre application, nous avons utilisé le système d'exploitation Microsoft Windows 7 sur lequel nous avons installé l'environnement de développement de la plateforme Qt et principalement Qt Creator (version 4.7).

4.2 Justification du choix technologiques

Nous avons choisi d'utiliser la plateforme Qt pour plusieurs raisons. Et parmi ces raisons nous pouvons citer les suivant :

- Un seul code source peut être compilé pour plusieurs système d'exploitation
- Réduction du temps de développement
- Profitez d'une véritable indépendance du système d'exploitation
- Qt est activement maintenu et développé pour soutenir toutes les nouvelles variantes du système d'exploitation grand public
- Qt augmente la productivité des développeurs en faisant programmation C++ plus rapide, plus facile et plus intuitive
- Programmation de l'interface graphique facile (par rapport à d'autres outils)
- Conteneurs de classes simples
- Environnement de développement (Qt Creator) simple et intuitive
- Excellente documentation avec des exemples utiles
- Facilité de l'internationalisation et de la localisation (aide à la traduction)
- Un bon support technique et une communauté très active

4.3 Aperçu sur le travail réalisé

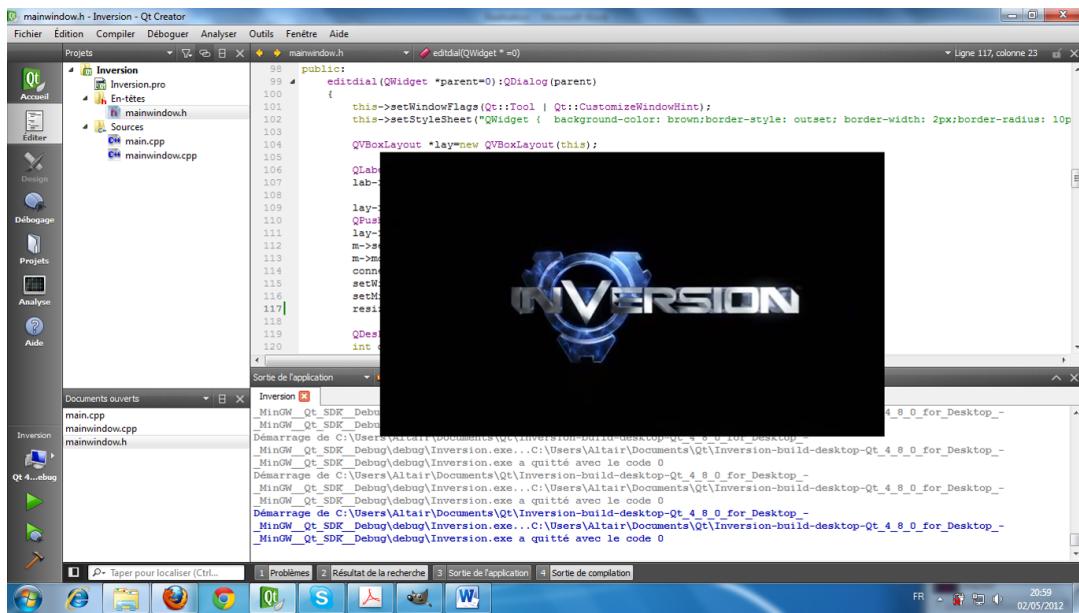


FIGURE 4.1 – Ecran d'accueil

La figure 4.1 montre l'écran d'accueil (le Splash screen) avec le logo du jeu d'inversion.

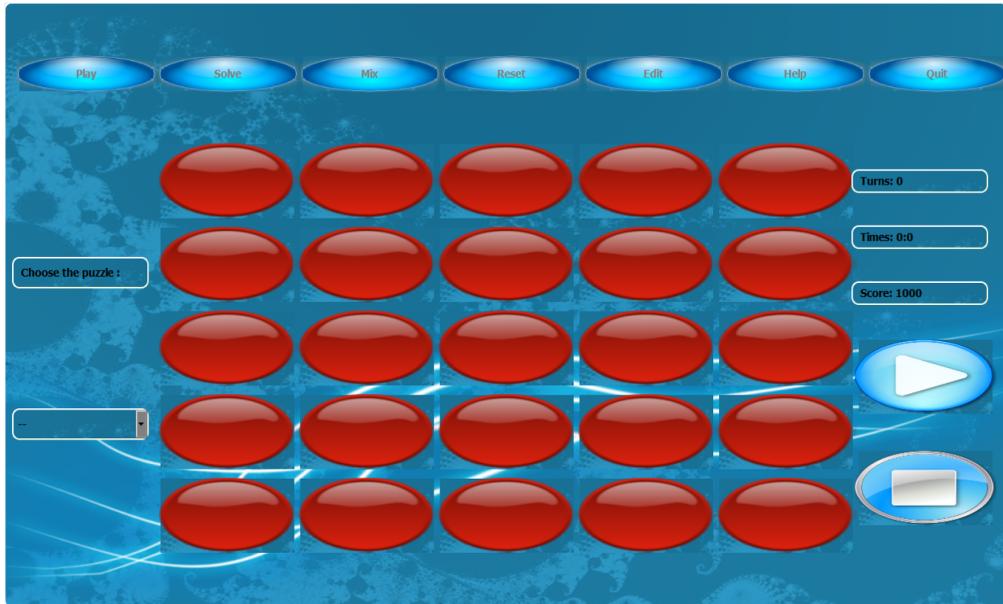


FIGURE 4.2 – Interface d'accueil

La figure 4.2 présente l'interface d'accueil de notre application. Cette interface est divisée en 4 parties :

- le menu avec des boutons de commande en haut de l'écran
- le menu de choix à gauche de l'écran
- le menu de temps et de score à droite de l'écran avec les boutons Play et Stop de la music de fond
- le menu du jeu qui contient la grille au milieu de l'écran

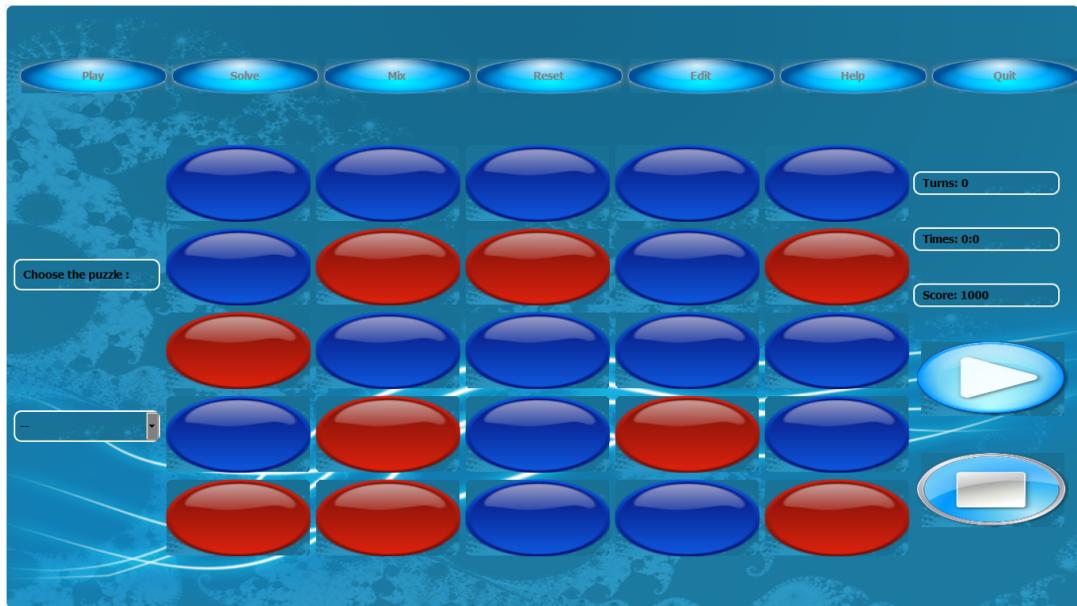


FIGURE 4.3 – Le bouton Mix

Le bouton Mix permet de générer une grille aléatoirement mais qui admet nécessairement une solution.

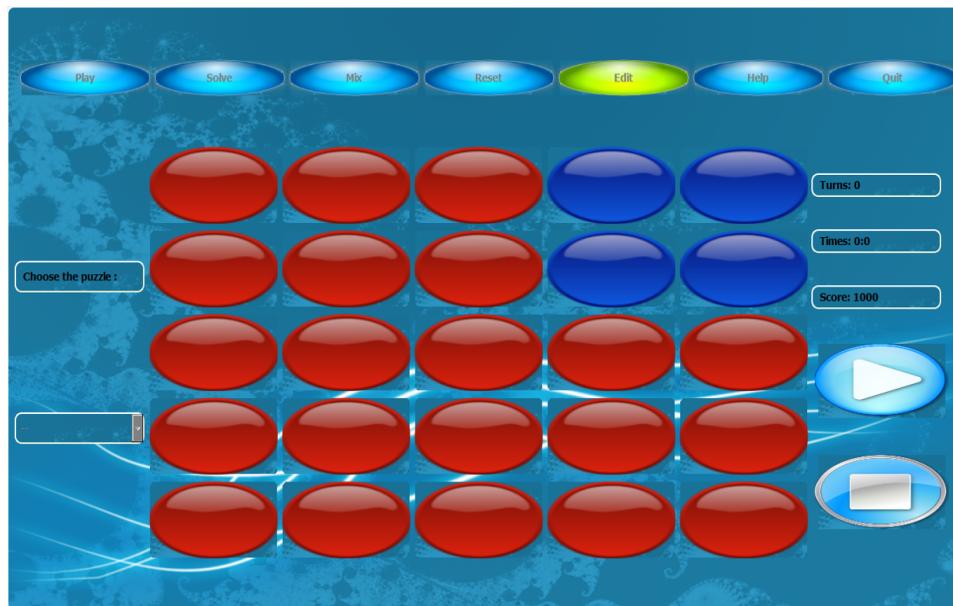


FIGURE 4.4 – Le bouton Edit

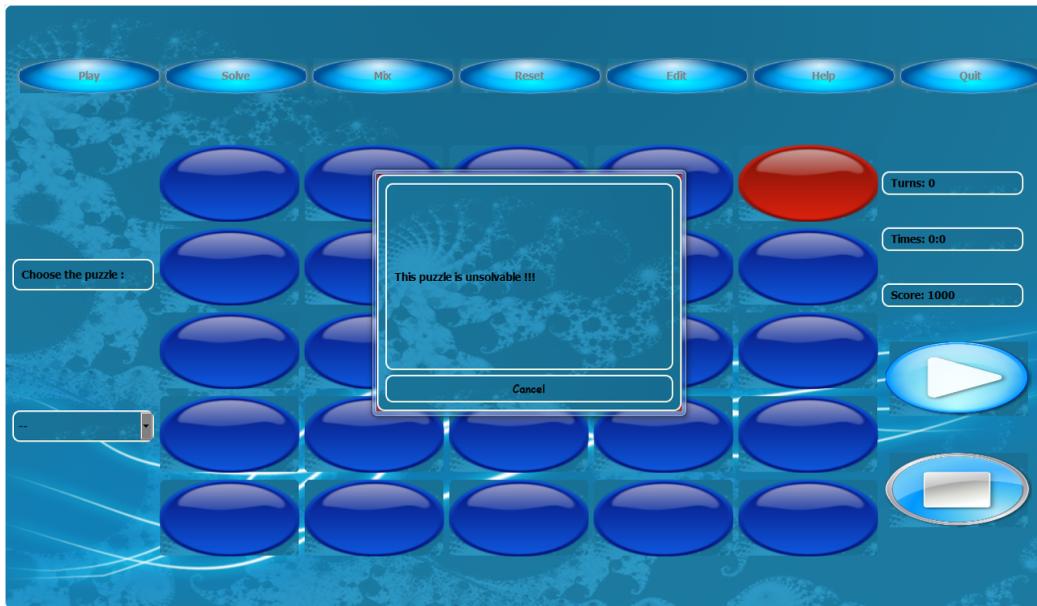


FIGURE 4.5 – Alerte : grille insoluble

Le bouton Edit permet de choisir une grille personnalisé entré par l'utilisateur (voir figure 4.4). Dans le cas où cette grille n'admet pas de solution une fenêtre de warning est affichée pour alerter l'utilisateur (voir figure 4.5).

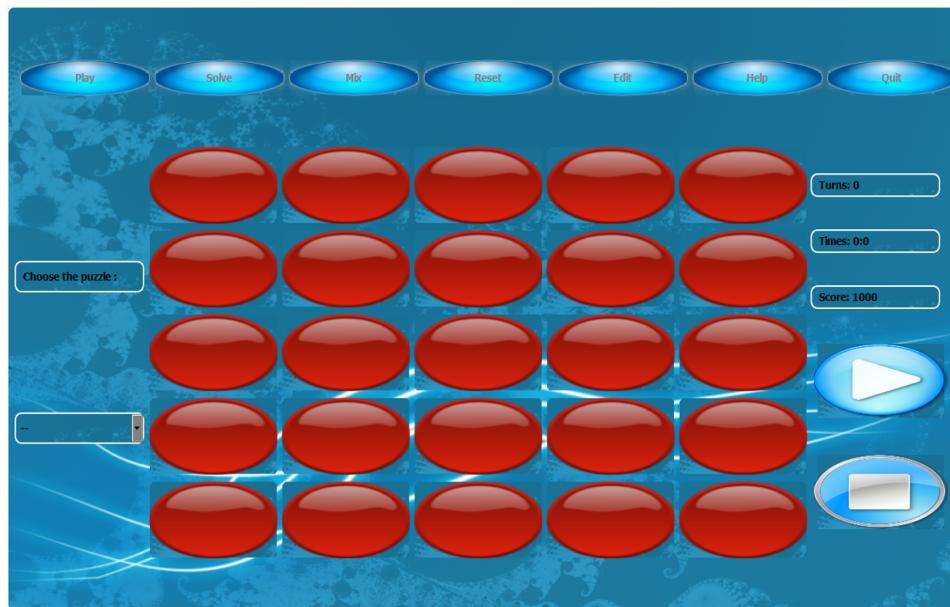


FIGURE 4.6 – Le bouton Reset

Ce bouton permet de réinitialiser la grille en mettant toute les cases à la couleur rouge.

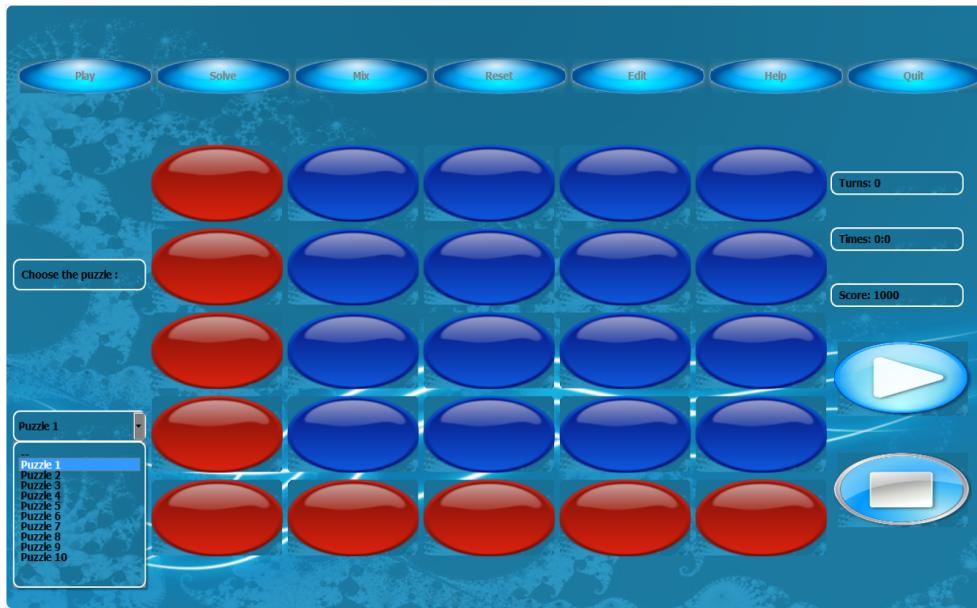


FIGURE 4.7 – Le menu de choix de grille

Le menu de choix de grille permet de choisir une grille prédéfinie. Ces grilles sont ordonnées par niveau de difficulté croissant de 1 à 10 (voir figure 4.7).

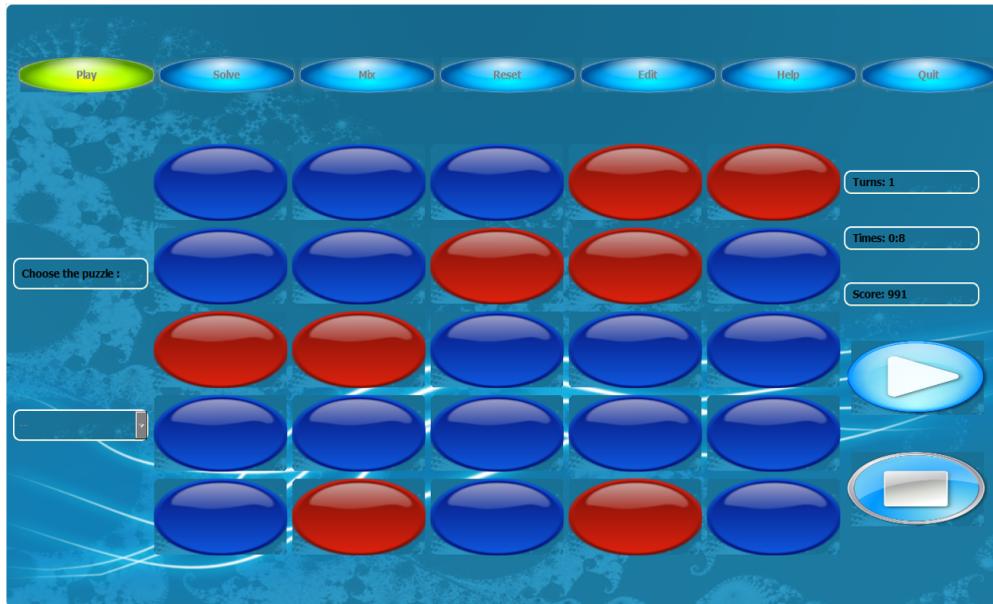


FIGURE 4.8 – Le bouton Play

Une fois la grille choisie par l'une des méthodes déjà cité. Le joueur peut commencer à jouer en respectant les règles du jeu d'inversion (voir figure 4.8).

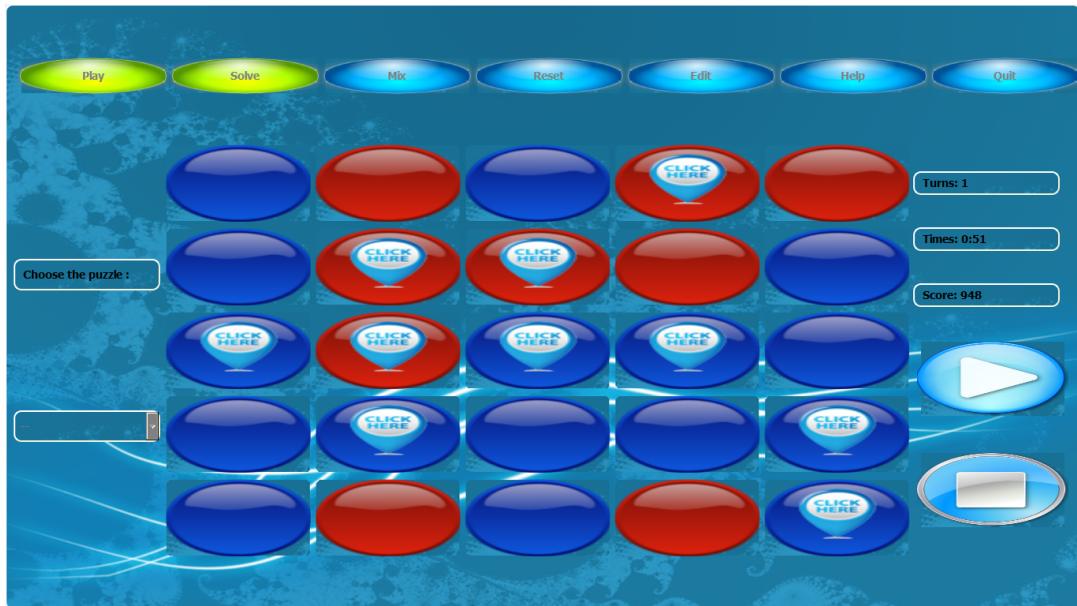


FIGURE 4.9 – Le bouton Solve

Si le joueur n'arrive pas à trouver une solution. Il peut appuyer sur le bouton Solve pour afficher la solution. La solution est affichée sous la forme d'une suite de boutons sur lesquels il faut appuyer pour résoudre la grille comme le montre la figure 4.9.



FIGURE 4.10 – Fenêtre Win

Une fois le joueur arrive à résoudre la grille avec succès une fenêtre indique au joueur qu'il a gagné la partie.



FIGURE 4.11 – Fenêtre Lose

Si le joueur n'arrive pas à résoudre la grille avant que le temps s'épuise, une fenêtre indiquant au joueur qu'il a perdu s'affiche.



FIGURE 4.12 – Le menu Help

Ce menu permet d'expliquer au nouveau joueur le principe du jeu (voir figure 4.12)

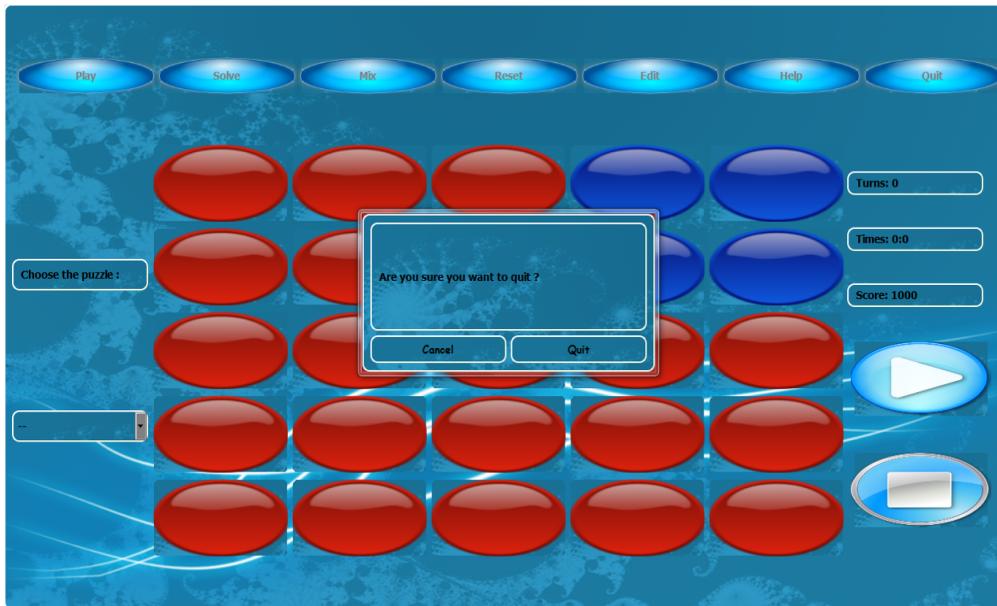


FIGURE 4.13 – Le menu Quit

Une fois le joueur a choisi de quitter le jeu, le menu permet d'alerter l'utilisateur s'il veut bien sortir comme le montre la figure 4.13.

Conclusion

Dans ce chapitre, nous avons présenté l'environnement de travail et exposé l'exécution de notre travail à travers des captures d'écran.

Conclusion Générale

Dans ce rapport, nous avons présenté toutes les étapes permettant la réalisation de notre jeu "Inversion". Dans le premier chapitre, nous avons présenté l'historique du jeu qui de l'année 1987. Ensuite, nous avons détaillé les besoins fonctionnels. Finalement, nous avons exposé la Conception détaillée du jeu. Des captures d'écrans ont été réalisées dans le chapitre "réalisation". Pour bien assimiler entre l'abstrait (solutions mathématiques, opérations sur les matrices) et l'interface graphique du jeu. Travailler sur le développement de notre jeu "Inversion" nous a été très enrichissant. Tout d'abord, l'utilisation des bibliothèques SDL Qt, a fallu que nous nous documentions plus là-dessus puisque les fonctionnalités basiques qu'on a apprises en classe concernant l'approche orientée objet ne suffisaient pas le moindre pour développer un jeu. Cette documentation, bien qu'elle nous ait appris beaucoup de temps, nous a été très bénéfique. Le jeu peut être beaucoup amélioré, on peut ajouter un fond musical, un compteur qui compte le nombre de frappe. Nous pourrions ajouter des niveaux à notre application. Dans ce cas, on atteindrait un autre niveau dès qu'on dépasse un certain score. Dans chaque nouveau niveau, une nouvelle couleur serait introduite. On peut aussi organiser les niveaux par la taille de la grille. Donc plus le niveau de difficulté augmente plus la taille de grille augmente.