Documents de spécification et de conception du projet C++
Jeu du 2048

Adrien DUFRAUX
Benjamin SCHULER
Amaury LACAVE
Benjamin DALLARD
Kamelya El ALAOUI

21 mai 2017

Table des matières

1 Introduction **2**

2 Les documents de spécification **4**

2.1 Les cas d'utilisations 4

2.2 Le diagramme de navigation 5

2.3 Spécifications des interfaces 5

2.4 Le diagramme de domaine 7

3 Les documents de conception **8**

3.1 Le diagramme de séquence système 8

3.2 Les diagrammes de séquence 9

3.2.1 Observation des deux IA 9

3.2.2 Jouer au 2048 9

3.3 Le diagramme de classes 10

3.4 Le diagramme de déploiement 11

4 Implémentation et tests **12**

4.1 Choix techniques 12

4.2 Tests unitaires 14

4.3 Guide d'utilisation 15

5 Conclusion **16**

Chapitre 1

Introduction

Le projet choisi a pour but de coder dans un premier temps le jeu du 2048. 2048 est un jeu vidéo de type puzzle conçu en mars 2014 par le développeur indépendant italien Gabriele Cirulli et publié en ligne le 9 mars 2014.

Le but du jeu est de faire glisser des tuiles sur une grille (même idée que le "taquin"), pour combiner les tuiles de mêmes valeurs et créer ainsi une tuile portant le nombre 2048. Le joueur peut toutefois continuer à jouer après cet objectif atteint pour faire le meilleur score possible.

2048 se joue sur une grille de 4x4 cases, avec des tuiles de couleurs et de valeurs variées (mais toujours des puissances de deux) qui peuvent être déplacées avec une animation de type "scrolling" quand le joueur appuie sur les touches fléchées de son clavier(ou avec la fonction tactile sur tablettes et smartphones).
Le gameplay du jeu repose ainsi sur l'utilisation des touches fléchées du clavier (ou de la fonction tactile sur tablettes et smartphones) pour déplacer les tuiles vers la gauche, la droite, le haut ou le bas. Lors d'un mouvement, l'ensemble des tuiles du plateau sont déplacées dans la même direction jusqu'à rencontrer les bords du plateau ou une autre tuile sur leur chemin. Si deux tuiles, ayant le même nombre, entrent en collision durant le mouvement, elles fusionnent en une nouvelle tuile de valeur double (par ex. : deux tuiles de valeur 2 donnent une tuile de valeur 4). À chaque mouvement, une tuile portant un 2 ou un 4 apparaît dans une case vide de manière aléatoire.
Le jeu, simple au début, se complexifie de plus en plus, du fait du manque de place pour faire bouger les tuiles, et des erreurs de manipulation possibles, pouvant entraîner un blocage des tuiles et donc la fin du jeu à plus ou moins long terme, selon la réflexion et lhabileté du joueur.
La partie est gagnée lorsqu'une tuile portant la valeur 2048 apparaît sur la grille, d'où le nom du jeu. On peut néanmoins continuer à jouer avec des tuiles de valeurs plus élevées (4096, 8192, etc.). La tuile maximum pouvant être atteinte est, en théorie, 131072 (ou 2^{17}) ; le score maximal possible est 3932156 ; le nombre maximum de déplacements est 1310387. Quand le joueur n'a plus de mouvement légaux (plus d'espaces vides ou de tuiles adjacentes avec la même valeur), le jeu se termine.

Dans un premier temps nous allons implémenter la structure du jeu afin de pouvoir jouer manuellement sur le terminal d'un ordinateur. Puis une version graphique utilisant les librairies Qt de C++ sera réalisée. La version graphique désirée doit être la plus proche possible de la vraie version même si on s'autorise des choix de couleurs peut être différents.
Après l'implémentation du jeu, on va s'intéresser aux méthodes de résolutions du 2048. Nous allons nous intéresser principalement à deux manières différentes :

- **Solveur explicite :** Ayant déjà joué au jeu, nous avons des stratégies qui nous permettent d'atteindre les meilleurs scores. Nous allons donc coder l'algorithme de résolution que nous réalisions lors de nos parties personnelles.
- **Réseaux de neurones :** Les réseaux de neurones sont pratiques pour réaliser des I.A car grâce aux données fournies, le réseau va "apprendre" à jouer et donc résoudre le problème.

Une fois les deux méthodes codées, il sera intéressant de comparer leurs performances respectives (temps de résolution, scores obtenus, ...).

Chapitre 2

Les documents de spécification

2.1 Les cas d'utilisations

Pour notre projet, nous avons accès à deux fonctionnalités principales :

— L'utilisateur peut jouer au jeu 2048 grâce aux flèches directionnelles.

— L'utilisateur peut laisser jouer une intelligence artificielle et l'observer. Nous ferons deux intelligences artificielles différentes. Une classique, appelée IA1, qui sera codée explicitement avec un algorithme classique. Une autre, IA2, s'appuiera sur des technologies de Machine learning. Le but est de pouvoir observer comment se comportent ces deux IA sur le jeu 2048.

— Dans le menu principal et dans le jeu nous avons aussi la possibilité de quitter le jeu.

Le diagramme de cas d'utilisation ci-dessous résume cela.

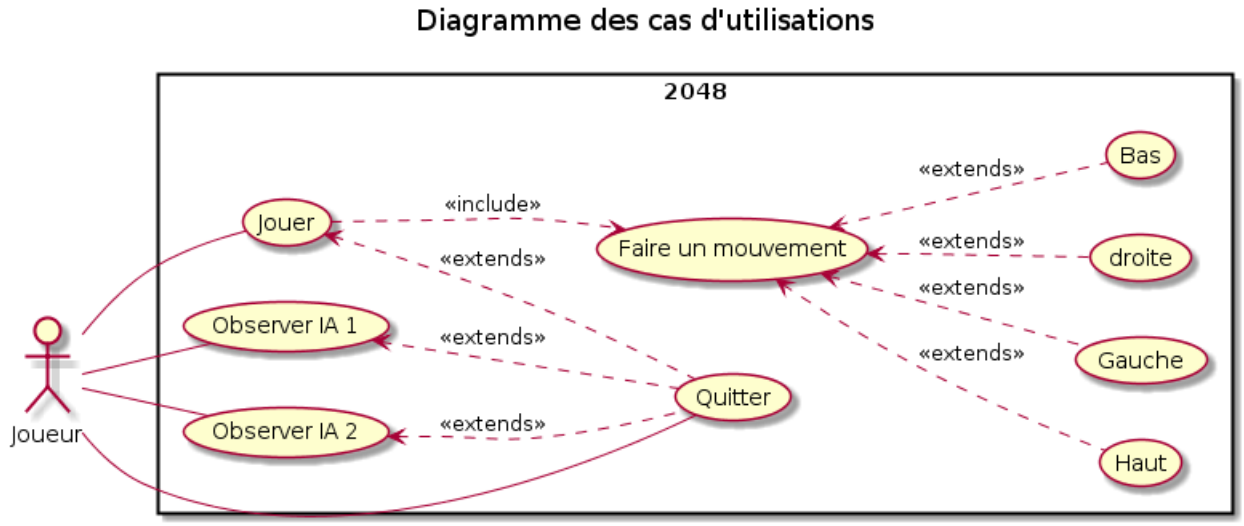

FIGURE 2.1 – Le diagramme des cas d'utilisations

2.2 Le diagramme de navigation

Ci-dessous se tient notre diagramme d'activité. L'utilisateur entre dans le menu principal et a le choix entre 4 actions : Jouer, observer IA1, observer IA2 et Quitter. Si l'option Jouer est choisie, l'utilisateur pourra alors faire des mouvements jusqu'à ce qu'il ai perdu. A tout moment, nous pouvons retourner dans le menu principal.

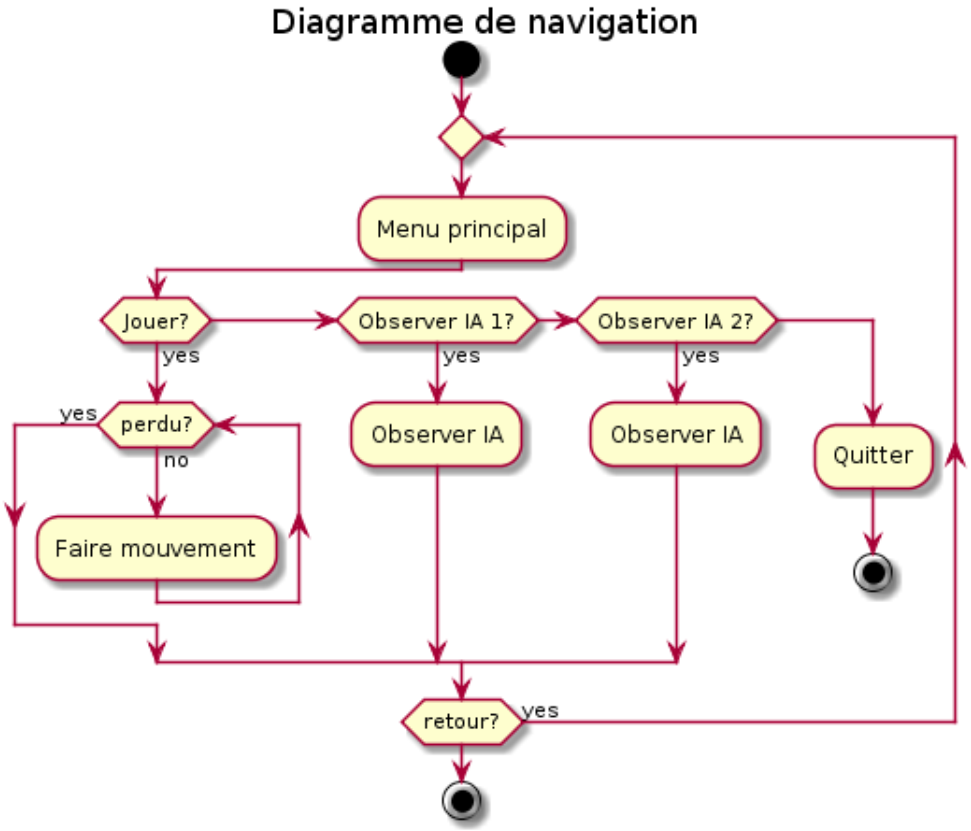

FIGURE 2.2 – Le diagramme de navigation

"Observer IA1" et "ObserverIA2" correspondent aux fonctionnalités expliqués précédemment.

2.3 Spécifications des interfaces

Nous aurons accès à deux fenêtres différentes pour notre jeu :

Le menu principal : Cette interface nous permet de choisir entre 4 boutons :

- Le bouton **Jouer** : Nous permet d'accéder à une utilisation normale du jeu.
- Le bouton **Observer IA1** : Nous permet d'accéder au jeu mais c'est lordinateur qui va jouer tout seul avec l'intelligence artificielle codée explicitement.
- Le bouton **Observer IA2** : Nous permet d'accéder au jeu mais c'est l'ordinateur qui va jouer tout seul avec l'intelligence artificielle qui s'appuie sur un réseau de neurones.
- Le bouton **Quitter** : Nous permet de quitter la fenêtre.

Sur la figure ci-dessous nous pouvons voir à quoi va ressembler l'interface du menu principal :

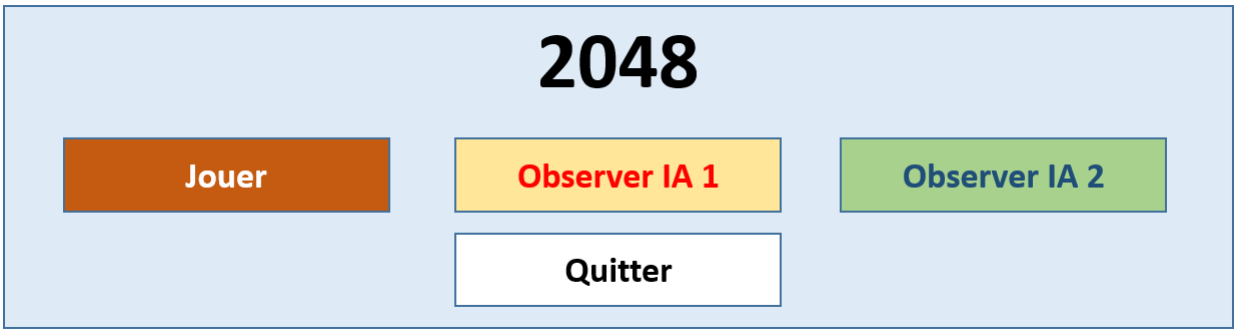

FIGURE 2.3 – Interface du menu principal

La fenêtre du 2048 : Cette fenêtre sera affichée quelque soit le choix de l'utilisateur au menu principal (sauf Quitter). On aura alors dans cette fenêtre : le jeu, le score du joueur en cours, le meilleurs score, qui est entrain de jouer, un bouton quitter et un bouton pour revenir au menu principal.

Ci-dessous, nous pouvons voir à quoi ressemblera le jeu au niveau de l'interface :

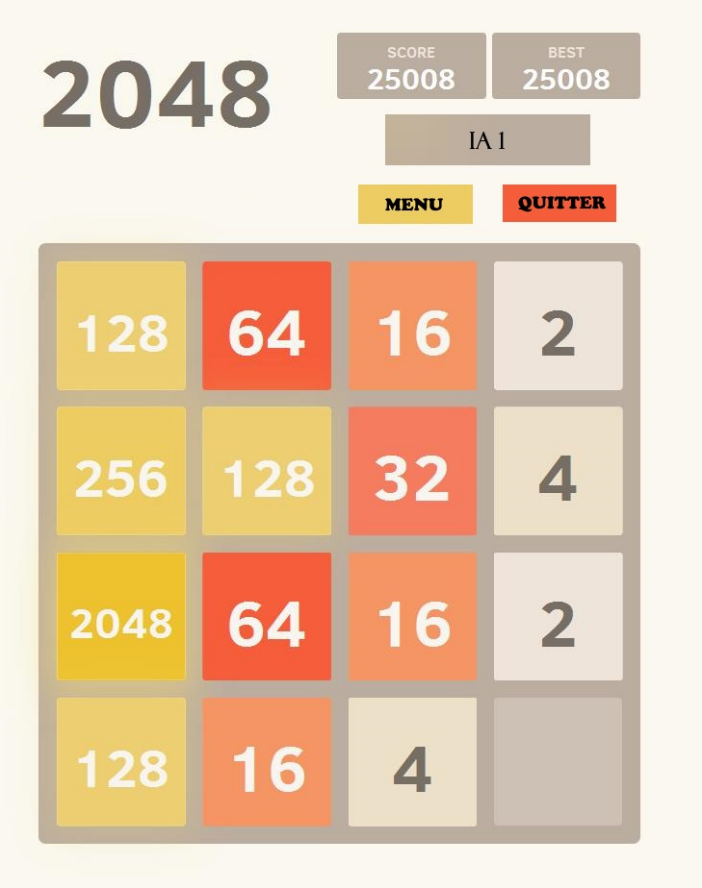

FIGURE 2.4 – Interface de la fenêtre de jeu

2.4 Le diagramme de domaine

Ci-dessous, notre diagramme de domaine. Il nous donne une idée de la structure du programme. Un joueur est donc soit un humain, soit une des deux intelligences artificielles. Le 2048 est composé de 16 cases et il est associé à un joueur.

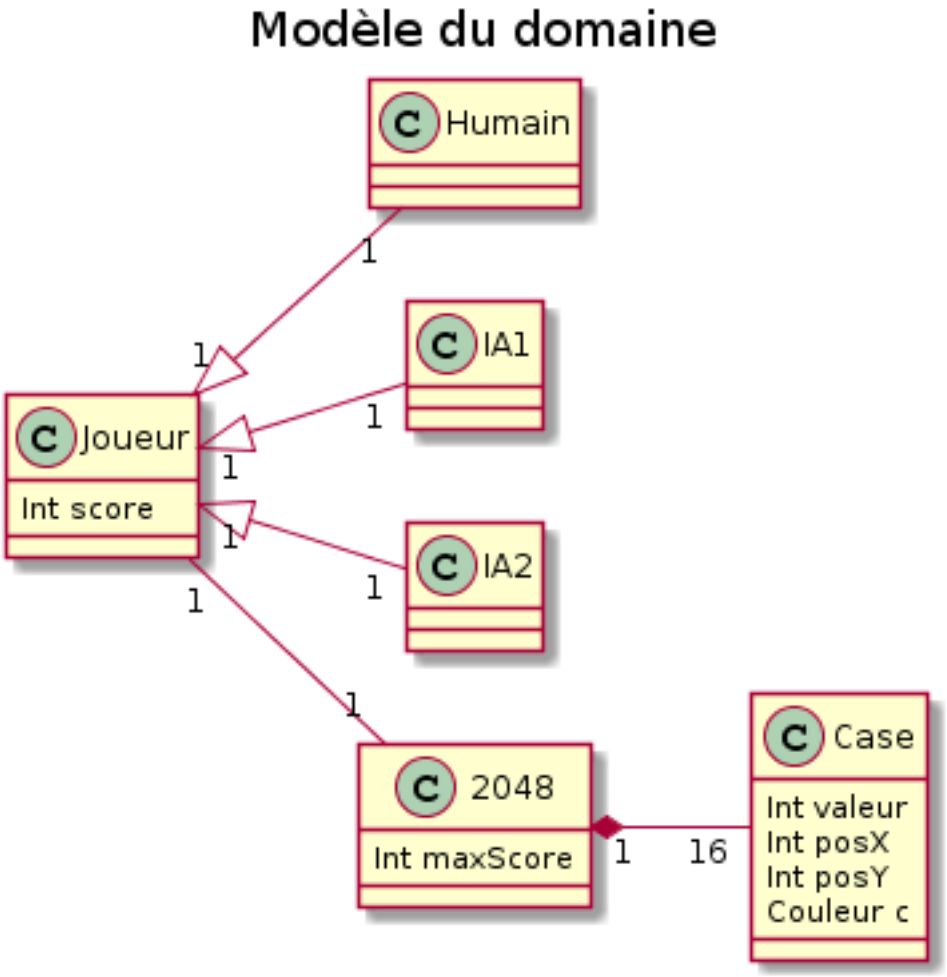

FIGURE 2.5 – Le diagramme de domaine

Chapitre 3

Les documents de conception

Cette section détaille les choix de conception qui on été pris. Nous présenterons nos choix avec les différents diagrammes de conception classiques.

3.1 Le diagramme de séquence système

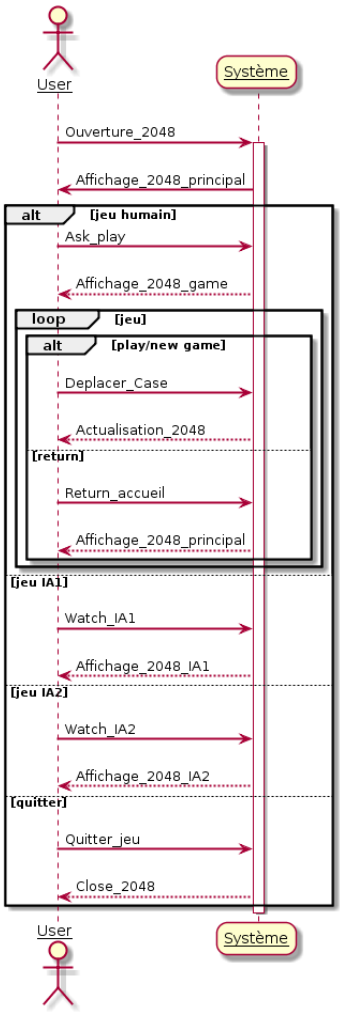

FIGURE 3.1 – Diagramme de séquence système

Nous pouvons voir ci-contre notre diagramme de séquence système. Nous pouvons voir les interactions entre l'utilisateur et le Système. Nous pouvons voir que l'utilisateur a le choix entre les quatre options de notre fenêtre de menu : Jouer, Quitter, observer IA1 et observer IA2.

Ce diagramme présente ainsi les différentes interactions possibles entre l'utilisateur et notre programme, et les prochains diagrammes de séquence que nous allons introduire vont détailler les différents scénarii possibles.

3.2 Les diagrammes de séquence

Nous allons décrire les différents scénarii avec des diagrammes de séquence.

3.2.1 Observation des deux IA

Ci-dessous, les deux diagrammes de séquence qui décrivent l'observation des deux IA.

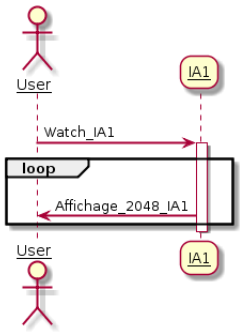

FIGURE 3.2 – Observation de l'IA1

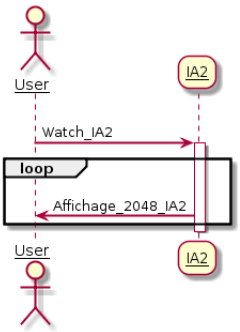

FIGURE 3.3 – Observation de l'IA2

3.2.2 Jouer au 2048

Ci-dessous, nous pouvons observer un utilisateur qui joue au 2048. Il utilise les flèches directionnelles jusqu'à ce qu'il perde la partie, ou décide de quitter le jeu. Les cases du jeu de déplacent à chaque fois et le score se met à jour.

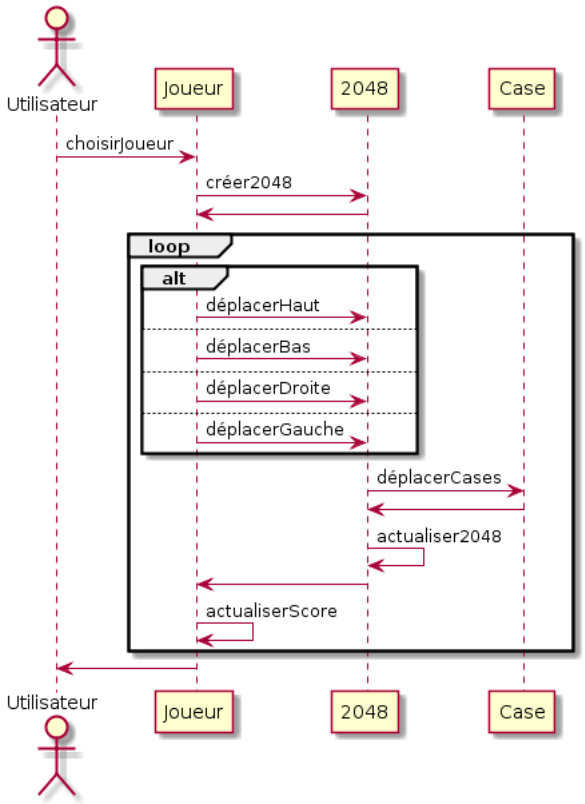

FIGURE 3.4 – Diagramme de séquence pour jouer au 2048

3.3 Le diagramme de classes

Nous allons maintenant présenter le diagramme de classes, qui nous permet de structurer la conception de notre projet et de pouvoir procéder à son implémentation.

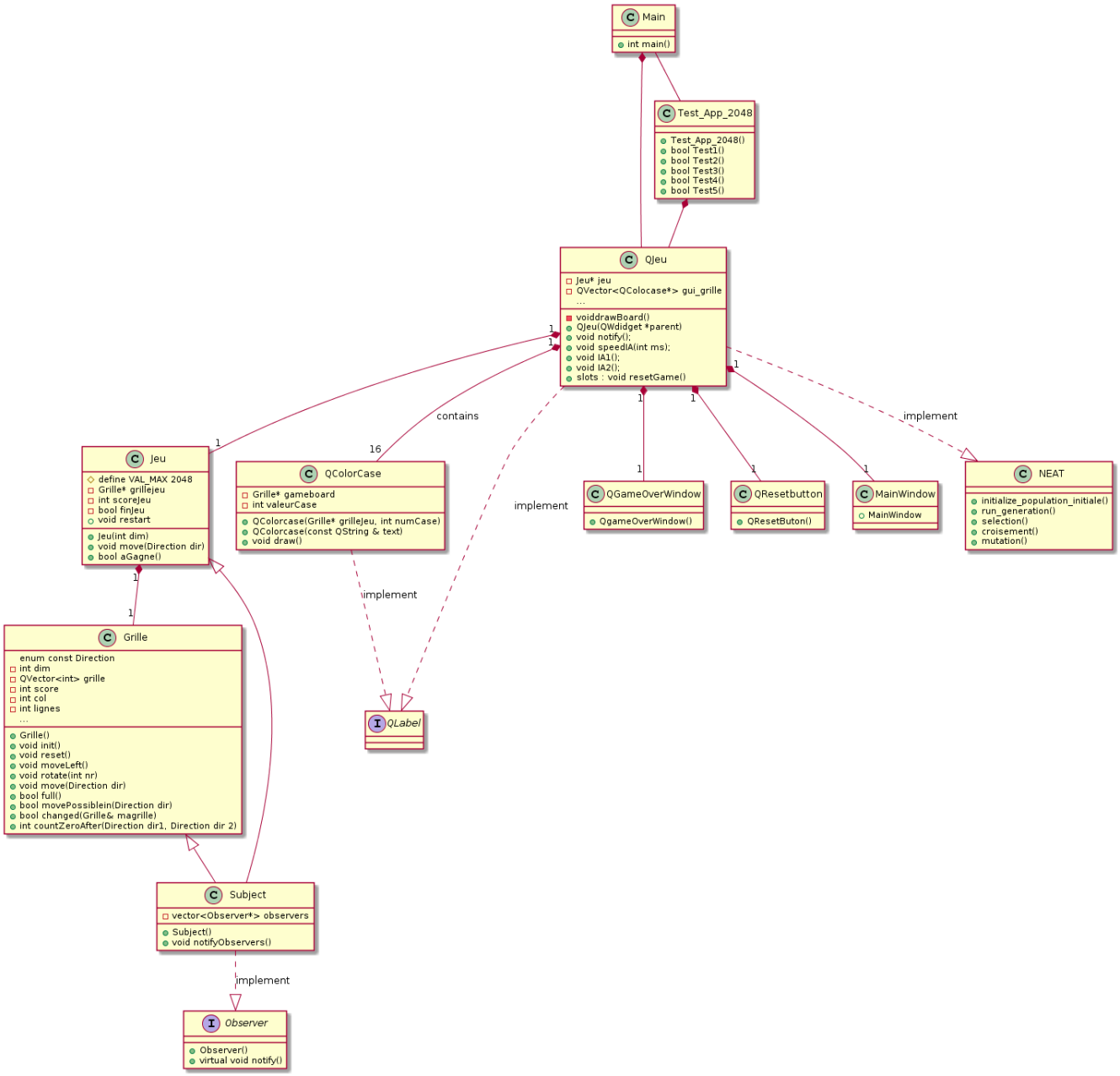

FIGURE 3.5 – Le diagramme de classes

C'est dans la classe Jeu que la valeur pour gagner est défini à 2048.
Cette dernière admet tout le corps du jeu sans la partie graphique.

Les classes dont la première lettre est un Q sont toutes les classes qui vont permettre l'affichage d'une interface graphique sous Qt.

La classe IA1 correspond à la méthode de résolution du 2048 par application d'un schéma tactique observé par la pratique des joueurs de 2048.

La classe IA2 correspond cette fois-ci à la résolution du 2048 à l'aide de réseaux de neurones. Elle implémente donc la classe NEAT.

La classe NEAT vient d'une bibliothèque de Machine learning. Elle va nous permettre de créer, dentraîner et d'utiliser un réseau de neurones. Elle mélange les technologie de machine learning et de réseau de neurones. Nous expliquerons cela plus en détail dans la partie choix techniques.

Le détail des classes sera donné dans la partie implémentation afin de comprendre plus en détail nos choix de classes.

3.4 Le diagramme de déploiement

Ci-dessous, nous avons notre architecture de fichier en package. Cette architecture nous permet de séparer le jeu en deux grandes partie : le corps du jeu et l'interface graphique. Cela permet de coder dans un premier temps le corps du jeu pour avoir un 2048 fonctionnel puis l'interface graphique vient après pour obtenir un jeu graphique dont le style est décidé par les préférences des programmeurs.

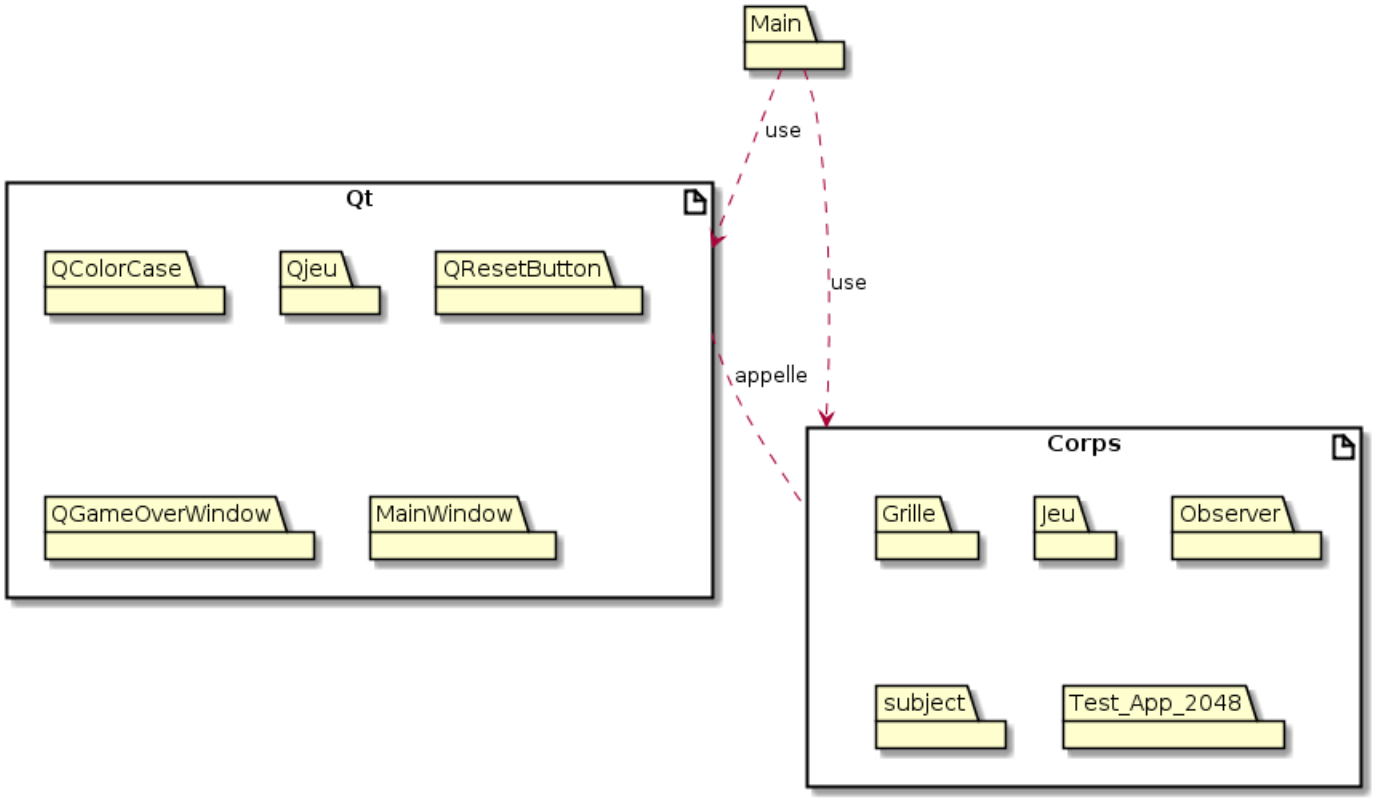

FIGURE 3.6 – Le diagramme de déploiement

Chapitre 4

Implémentation et tests

4.1 Choix techniques

Afin de mener à bien notre projet, nous avons choisi d'utiliser, en plus du langage de programmation $C++$, la bibliothèque logicielle multiplateforme Qt pour réaliser l'interface graphique.

Nous allons maintenant expliquer nos choix de conception et d'implémentation.
Au cours d'une première réflexion nous avions décidé de faire un tableau à deux dimensions de cases. Puis opérerions sur les cases et le tableau pour effectuer les déplacements qu'impliquent ce jeu.
En se posant un peu plus sur la nécessité de raisonner comme cela, nous nous sommes aperçu qu'il n'était pas nécessaire de raisonner sur un tableau de cases mais que seul un tableau nous serait nécessaire. En effet, une case vide correspondrai à une valeur 0 dans le tableau et les autres valeurs non nulles représenteraient bien des cases. Il suffit donc de bien spécifier le code couleur à appliquer lors de l'implémentation de l'interface graphique Qt pour voir une "grille de cases".
De plus, il n'est pas nécessaire de travailler sur un tableau à deux dimensions alors qu'un tableau simple de 16 cases suffit pour un jeu en 4x4. Cela facilite l'implémentation des méthodes et nous permet d'éviter de faire des doubles boucles pour explorer le tableau. Il faut juste penser à une astuce mathématique pour afficher le tableau comme on le désire : l'utilisation du *modulo* afin de revenir à la ligne quand il faut selon les dimensions du jeu.
Travailler sur un tableau simple à l'avantage d'être plus rapide lors de l'exécution de certaines méthodes même si dans notre cas l'avantage n'est pas appréciable au vu de la "simplicité" des méthodes.
Une dernière remarque à faire concerne le déplacement dans la grille. On a quatre directions possibles donc quatre fonctions de déplacements différentes mais on peut en faite coder seulement deux méthodes qui vont nous permettre d'effectuer tout les déplacements : $moveLeft()$ et $rotate(intnr)$.
La première méthode est explicite, elle code le déplacement vers la gauche de la grille et la seconde permet d'effectuer une rotation dans le sens horaire de la grille autant de fois que l'entier nr. En effet, si l'on effectue une double rotation de la grille, qu'on effectue la fonction $moveLeft()$ et qu'on ré effectue une double rotation, alors on aura réaliser un déplacement vers la droite. Pa conséquent les fonctions de déplacement nécessite nr = 1,2 ou 3 rotation(s) (bas, droite, haut), un mouvement vers la gauche avec $moveLeft()$ puis "4-nr" rotations pour se remettre dans le référentiel initial.

Parlons maintenant du corps du jeu avant d'expliquer l'implémentation graphique sur Qt.
Nous avons donc besoin d'un tableau de 16 entiers qui représentera la grille et cette grille aura des dimensions, un score et pourra bouger sur les 4 directions possibles.
En vu de l'implémentation de l'interface graphique, nous avons préférer l'utilisation de $QVector < int >$ pour la grille de jeu et des entiers pour toutes les caractéristiques de la grille de jeu.
Nous avons également défini le type *Direction* qui est une énumération comportant les quatre directions : enum Direction UP, DOWN, LEFT, RIGHT . La classe Grille va donc permettre de réaliser les opérations de déplacement sur la grille. On va vérifier que le déplacement est possible grâce à la fonction $changed(Grille\&magrille)$

qui va tester si deux grilles sont identiques. D'autres fonctions regarderont si la grille est pleine, si le déplacement dans une direction est possible, l'affichage de la grille et une fonction $counZeroAfter(Directiondir1, Directiondir2)$ qui permet de savoir combien on aura de cases vides après le déplacement selon dir1 puis dir2. Cette fonction sera utiliser pour l'IA explicite.
La classe Grille étant codé, on crée une classe Jeu qui possédera une Grille grille qui sera la grille de jeu, un booléen représentant la fin du jeu, et le score du jeu en cours d'exécution.
Au niveau des méthodes, on notera la surcharge de la méthode $move(Directiondir)$ également présente dans la classe Grille qui permet de déplacer la grille de jeu si cela est possible. En cas de non possibilité d'effectuer un déplacement peu importe la direction, le jeu est fini (bool finJeu = true) et on peut effectuer la méthode $restart()$ pour effectuer une nouvelle partie.
Deux classes Observer, Subject sont des classes "interfaces" qui permettent l'interaction graphique sur Qt, notamment avec la gestion des slots et connection qui permettent de changer de fenêtre ou d'en ouvrir d'autres.

Maintenant décrivons la partie interface graphique.
La première classe Qt réaliser est QColorcase qui va permettre de donner le style propre du jeu en fonction de la valeur donné. Il faut donc spécifier la couleur pour chaque valeur que peut prendre une case. Nous avons volontairement changé de couleurs par rapport au eu initial afin de s'approprier le projet et de faire une simple imitation.
Les classes QMainWindow, QResestBouton et QGameOverWindow sont les classes qui permettent d'afficher les fenêtre principale, en cas de fin de jeu et pour rejouer le jeu.
La classe réellement importante de l'interface va être QJeu. Le constructeur de cette classe va permettre de définir la fenêtre de jeu et dessiner en conséquence de la grille de jeu initial.
Pour avoir ce style de grille de jeu on utilise l'objet QGridLayout et les Qwidget afin d'obtenir un style cohérent avec le jeu, défini auparavant dans QColorcase.
La même astuce que dans la méthode d'affichage de la grille de jeu va permettre de disposer les valeurs dans les bons "widget" du "gridLayout".
Ensuite on définit les événements occasionner lors de l'utilisation des flèches du clavier afin de bouger correctement et pour finir on aura bien sur la procédure de dessin $drawboard()$ qui va dessiner à l'écran la grille de jeu après chaque déplacement.

la classe main va permettre de lancer le jeu et d'afficher les choix disponibles pour l'utilisateur : jouer(solo), regarder l'IA1 ou l'IA2 résoudre le 2048.

IA explicite Parlons de l'IA explicite qui utilise un algorithme codé par nous même pour résoudre le jeu.
Au début une classe à part entière devait être faite pour cette IA mais il est apparu inutile de le faire et à la place de créer une procédure dans la classe QJeu qui effectue le rôle de l'IA lorsqu'on appuie sur la touche *enter* du clavier.
L'algorithme utiliser est très simple et réside sur le principe qu'il y a une direction "interdite" ou à utiliser en dernier recours. Dans notre cas cette direction est "UP". On part du principe qu'on veut toujours aller vers le bas si c'est possible c'est possible. A contrario, si c'est pas possible il faut savoir quelle direction choisir entre "LEFT" et "RIGHT". Nous allons donc enfin utiliser la fonction $countZeroAfter(Directiondir1, Directiondir2)$ afin de compter le nombre de case vide qu'on aura après "LEFT,DOWN" et "RIGHT,DOWN" et privilégier la direction qui occasionnera potentiellement le plus de cases vides. Potentiellement car une part d'aléatoire devrait être pris en compte car un 2 (90%) ou un 4 (10%) apparaît aléatoirement dans les cases vides après chaque déplacement.
En cas d'égalité, nous privilégions la gauche à la droite et si aucun autre mouvement n'est disponible nous effectuons un mouvement vers le haut.
La fonction $speedIA(intms)$ permet d'actualiser la grille de jeu pour visualiser les déplacements réaliser par l'IA.
Cette approche d'IA est facile à comprendre et à implémenter mais donne des résultats peu satisfaisant dans le sens où nous atteignons 7 fois sur 10 le score de 512, 2 fois sur 10 le score de 1024 et 1 fois sur 10 l score de 256.
La réalisation de l'iA était un choix ambitieux et il nous aurait fallu plus de temps pour peaufiner cette IA afin qu'elle atteigne 2048.

IA implicite : Nous avons essayé d'implémenter une autre IA qui se base sur la technologie des réseau de neurones. Le problème est un peu différent des techniques de machines learning classique car nous ne disposons pas de données au départ. Il est sûrement possible d'en générer avec le jeu mais nous nous sommes orientés vers une autre approche. Après quelques recherche, nous avons utilisé l'outil NEAT (Neuroevolution of augmenting topologies) qui a été développé par Ken Stanley en 2002 à l'université du texas à Austin.

C'est en fait un mélange de deux technologies : Les algorithmes génétiques et le réseau de neurones artificiels. Le principe est que nous partons d'une population de réseau de neurones qui sont générés aléatoirement au début. Nous définissons juste la structure de départ qui doit être minimale. Comme nous avons 16 cases pour le 2048 et 4 directions possibles. Nous créons des réseaux de neurones avec 16 entrées et 4 sorties. L'algorithme effectue alors les étapes de sélection, croisement et mutation qui sont classiques des algorithmes génétiques. Ce que l'algorithme cherche à obtenir est un réseau de neurones qui maximise un certain critère. Lors de ces étapes qui composent une génération de lalgorithme génétique, les réseaux peuvent être modifiés de plusieurs façon. La structure même du réseau peut être modifiée (La topologie) : Ainsi, de nouveaux neurones peuvent apparaître ou disparaître. Les poids entre les neurones peuvent aussi être modifiés. Au fur et à mesure que l'algorithme tourne, il doit pouvoir sélectionner les individus et créer de nouveaux réseaux qui vont être composés de ce qu'il y a de meilleurs dans les caractéristiques des réseaux de neurones parents.

Le critère à maximiser est ici le score qu'un réseau de neurones pourrait obtenir en jouant une partie. Pour éviter de s'en remettre qu'à la chance, nous faisons jouer à chaque réseau 10 parties de 2048 et nous calculons la moyenne des scores.

Nous trouvons cette approche très intéressante, et le 2048 nous sert de prétexte pour pouvoir l'essayer. Malheureusement, nous n'avons pas obtenu de résultats très concluants. Nous avons réussi à faire fonctionner l'outil grâce au manuel d'utilisation qui était fourni mais même au bout de 2000 générations, les réseaux obtenus ne pouvait pas faire plus d'un 256 lors d'une partie. Cette approche n'est peut être pas adaptée à ce type de problème. Quoi qu'il en soit, ce fut intéressant d'implémenter cette technologie.

Par conséquent nous avons implémenter plusieurs IA différentes autres que l'IA explicite définit plus haut. Dans le code, il y a possibilité de choisir celles qu'on veut (IA1, IA2, IA3 et IA) ilsuffit juste de spécifier dans la méthode $keyPressedEvent()$ quelles doivent être les IA lancées lorsqu'on appuie sur la touche "espace" ou "entrée" du clavier. Par défaut, "entrée" lance l'IA et "espace" lance l'IA3.

4.2 Tests unitaires

Pour les tests unitaires nous avons créé une classe Test_App_2048 qui va permettre de réaliser les principaux tests unitaires du jeu. Nous n'avons pas testés chacune des fonctions écrites mais nous avons choisi les fonctions principales du jeu et celles qui appellent beaucoup de fonctions essentielles.
En tout nous avons 5 "tests" dans cette classe qui vont tester les fonctions suivantes :

- création d'un jeu
- récupération du score et de la dimension
- remplissage d'une grille de jeu par un tableau d'entier
- mettre ou récupérer la valeur d'une case de la grille de jeu
- vérification du mouvement vers la gauche
- vérification de la rotation
- vérification sur une grille pleine pour savoir si on peut encore effectuer un mouvement
- vérification de la victoire du joueur

Voici la capture d'écran de lexécution des tests :

```
Démarrage de C:\Users\benjamin\Desktop\GM\QT_Project\build-2048_vB-Desktop_Qt_5_9_0_MinGW_32bit-Debug\debug\2048_vB.exe...
                    Batterie de tests unitaires sur le jeu 20480
 Test 1 : renvoie 1 si on recupere la bonne dimension, le bon score et fonction setValue fonctionnelle -->  1
 Test 2 : renvoie 1 si la fonction moveLeft est fonctionnelle -->                                          1
 Test 3 : renvoie 1 si la fonction rotate est fonctionnelle -->                                            1
 Test 4 : renvoie 1 si le test de la fonction aGagne() est valide  -->                                     1
 Test 5 : renvoie 1 si le test des fonctions isGameOver(), full() et movePossible() est valide -->         1
```

4.3 Guide d'utilisation

Afin d'utiliser notre jeu, exécutez le programme puis au menu principal, sélectionnez "Jouer". En effet, les autres boutons réservés aux IA ne sont pas fonctionnels, mais leur texte explique comment appeler une IA. Une fois sur la fenêtre de jeu, voici les trois manières de jouer :

- Joueur humain : Utilisez les flèches directionnelles (haut, bas, gauche et droite) pour déplacer les cases du 2048 dans la direction souhaitée. Nous rappelons que le but du jeu est d'assembler les cases de même valeur deux à deux afin de doubler la valeur de la case obtenue, et d'atteindre le plus gros score possible avant de ne plus pouvoir se déplacer.
- IA 1 : Comme indiqué sur le bouton du menu principal, appuyez sur la barre d'espace pour lancer l'IA et profitez du spectacle. Il faut cependant appuyer d'abord sur le bouton "jouer" puis ensuite une fois la grille apparue, presser le bouton espace.
- IA 2 : A l'instar de l'IA 1, il vous suffit de cliquer sur jouer puis d'appuyer sur la touche *Entree* et de regarder l'évolution du jeu.

Chapitre 5

Conclusion

L'objectif de notre projet était de développer un jeu 2048 accompagné de deux Intelligences Artificielles, une étant implémentée selon une stratégie connue permettant d'atteindre les meilleurs scores, et la seconde à l'aide d'un réseau de neurones, qui allait "apprendre" à bien jouer et enfin résoudre le jeu. Nous avons ainsi pu implémenter l'ensemble de ces aspects, en commençant par créer le code du jeu 2048, puis nous lui avons ajouté les IA décrites précédemment. Nous disposons donc d'un jeu composé d'un menu principal et de la fenêtre de jeu, où l'on peut soit jouer par nous-mêmes, soit regarder une IA essayant de résoudre le jeu.

Implémenter des IAs performantes étaient un projet ambitieux et nous ne sommes malheureusement pas parvenu à faire le réseaux de neurones attendus. L'autre IA fonctionne mais elle n'a pas encore réussi à atteindre plus de 1024. Plus de temps, nous aurait permis de la travailler plus et de réfléchir à une amélioration de celle-ci.

Nous aurions pu améliorer notre projet en y ajoutant un mode compétition, où deux jeux se dérouleraient en même temps sur la même fenêtre, et on lancerait une IA sur un des jeux, et on choisirait soit de jouer soit de lancer une autre IA sur le deuxième jeu. Ainsi on aurait pu comparer en temps réel qui est le meilleur joueur de 2048. Cependant, n'ayant pas prévu cela lors de la conception, et aussi par manque de temps, nous n'avons pas implémenté cette option.

Ce projet nous a permis de mettre en pratique les enseignements reçus sur le langage de programmation $C++$, et nous avons pu découvrir et prendre en main la bibliothèque logicielle Qt, qui nous a permis d'obtenir efficacement notre interface graphique.

Pour conclure, nous avons obtenu un 2048 jouable et des IA explicites opérationnelles (mais pas optimales), et avons ainsi validé les objectifs principaux de notre projet. De plus, nous avons pu concrétiser et améliorer nos connaissance sur la programmation en $C++$ et découvrir Qt, et nous allons probablement continuer le développement de notre jeu. Voila pour le moment le jeu obtenu ci dessous avec le meilleur coup de l'IA explicite.

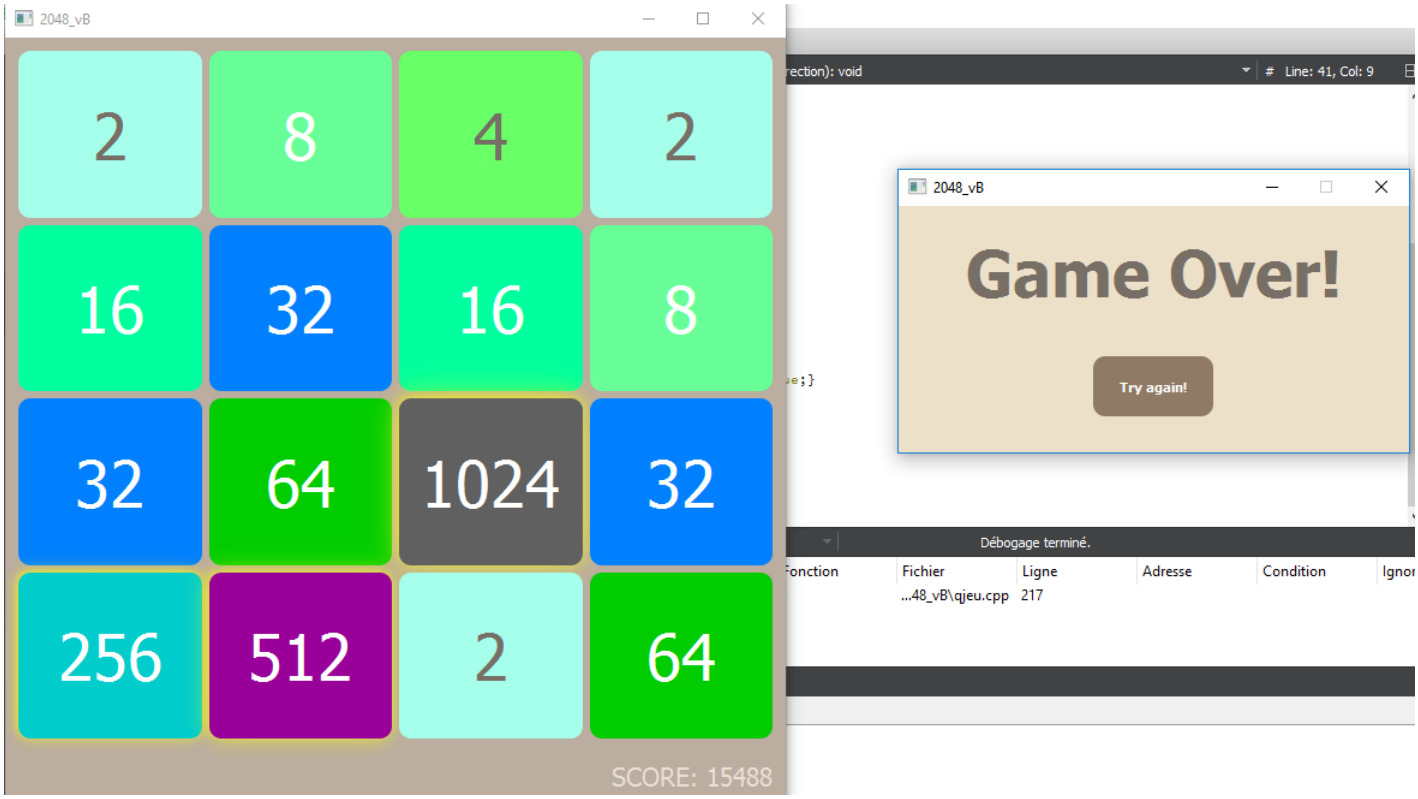

FIGURE 5.1 – Le meilleur coup obtenu avec l'IA