MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE (TOGO)

Université de Lomé | CENTRE INFORMATIQUE ET DE CALCUL (CIC)

01 BP 1515 Lomé 01 – TOGO
Tel : (0028) 22 25 33 29 Fax : (0028) 22 21 85 95
Site web : www.univ-lome.tg

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR, DE LA RECHERCHE ET DE L'INNOVATION (FRANCE)

utbm université de technologie Belfort-Montbéliard

UTBM – 90010 BELFORT CEDEX
Tel : +33 (0) 3 84 58 30 00 Fax : +33 (0) 3 84 58 30 30
Site web : www.utbm.fr

MASTER INFORMATIQUE

Spécialité : **Génie Logiciel**

TRAVAUX PRATIQUE DU COURS DE PROGRAMMATION ORIENTEE OBJET AVANCÉE

Thème :

CONCEPTION ET IMPLEMENTATION DU JEU « DEMINEUR »

Effectué par :

KUDEABO Kossi Frédéric Aldo

WOAGOU Pouguinipo Daniel

Responsable du cours :

Amir Hajjam El Hassani,
Responsable de la formation d'ingénieurs sous statut d'apprenti

Sommaire

Introduction

La pratique de la programmation orientée objet passe par une approche radicalement différente des méthodes de programmation traditionnelles. Les langages objets font des programmeurs des metteurs en scène d'un jeu collectif où chaque objet-acteur (à qui on doit attribuer un rôle bien précis) contribue au succès du scénario. Le cours de Programmation Orientée Objet Avancée a pour but de nous doter des compétences pour mieux cerner ce jeu à travers le langage le C++.

Ce projet, en s'inscrivant dans la continuité du cours, représente deux challenges pour nous : d'une part il nous offre l'opportunité d'un premier contact avec le développement graphique sous C++ et d'autre part c'est une occasion pour nous d'élargir nos champs de compétence sur le langage C++.

Ce document a pour objet de fournir un rapport détaillé sur les étapes suivies pour la conception du projet. Il est subdivisé en quatre (04) parties : la première se chargera de l'étude du cahier des charges, la deuxième sera consacrée l'analyse et à la conception en UML, la troisième partie présentera l'implémentation faite en C++ et la dernière servira de guide pour les futurs utilisateurs.

I. Etude des besoins

La réalisation de ce jeu nécessite la conception d'un système objet, souple, fiable et facile à maintenir. Pour ce faire il nous faut recenser les contraintes fonctionnelles et non fonctionnelles du jeu, analyser les scénarios possibles, les performances attendues pour ce qui est de la facilité d'utilisation et en déduire une conception adéquate.

A. Contraintes

1. Contraintes fonctionnelles

Comme décrit dans le sujet, les fonctionnalités du jeu du démineur sont pareilles que celles présentes dans les systèmes d'exploitation Windows. Pour un souci de lisibilité et de concision nous ne reprendrons pas la description des fonctionnalités.

2. Contraintes non fonctionnelles

Pour assurer une expérience de jeu agréable aux utilisateurs, nous développeront (à l'aide de l'environnement de développement intégré) des interfaces ergonomiques et intuitive,

B. (Scénarios)

C. Les performances attendues

Notre version du jeu démineur devra tout d'abord être facile à installer même pour un néophyte, légère et robuste. Elle sera fournie avec un manuel d'installation et un mode d'utilisation (fourni à la partie IV de ce document).

II. Analyse et conception

À la suite de l'étude des besoins, nous avons (conçu) et modélisé notre système en se basant sur les différents principes, règles et patrons conception qui régissent la programmation orientée objet. Ci-dessous les différents diagrammes représentant notre système.

A. Diagramme des cas d'utilisation

Les cas d'utilisation sont une représentation des différentes fonctionnalités qu'un système possède ainsi que les acteurs pouvant y avoir accès. Dans notre cas, il s'agit de schématiser les fonctionnalités de notre jeu.

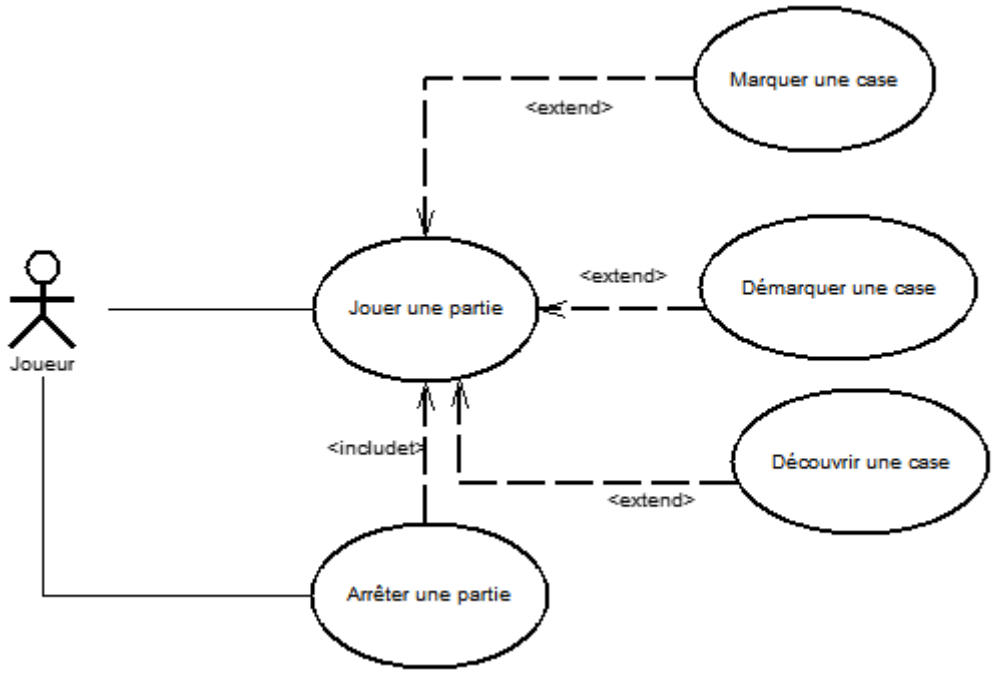


Figure 1: Diagramme des cas d'utilisation

B. Diagramme des classes

Une bonne conception objet passe par une bonne définition de la structure des objets à manipuler. Pour ce faire, il revient de concevoir des classes répondant de manière spécifique au besoin du système, et facilitant l'évolutivité et une bonne maintenance. Pour optimiser le schéma, toutes les méthodes de classe ne seront pas listées sur le diagramme suivant. Une description plus exhaustive des différentes classes sera donnée.

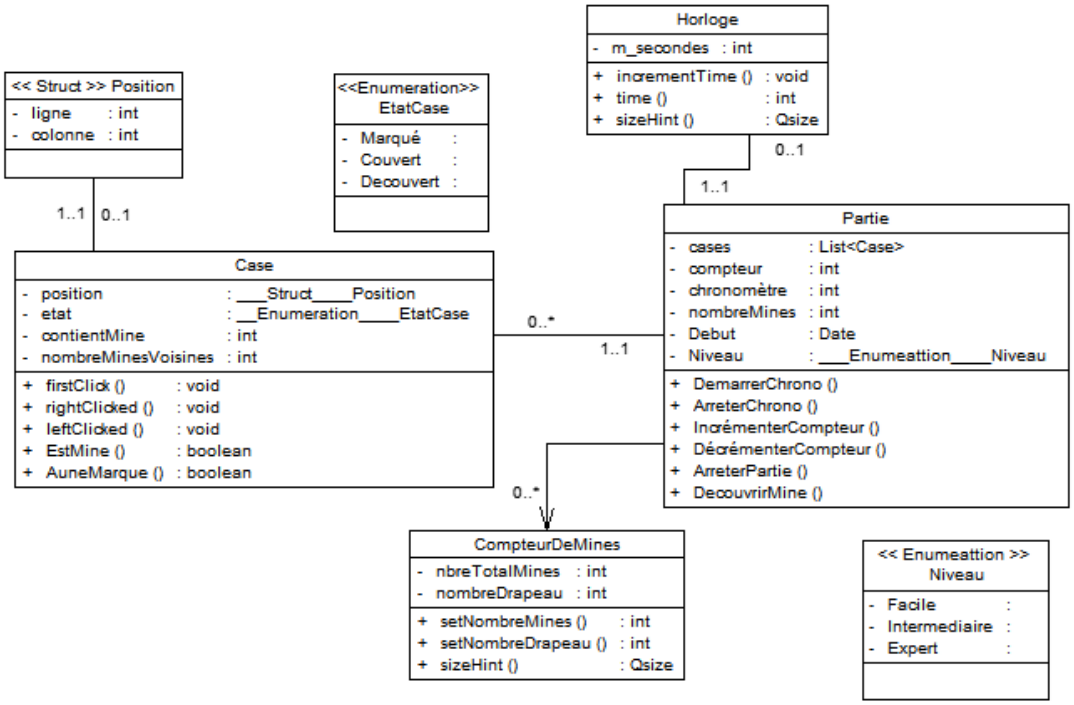


Figure 2: Diagramme des classes

C. Diagramme dynamiques

1. Diagramme de séquence

Permettant de schématiser de manière séquentielle les interactions qu'il peut y avoir entre le joueur et notre jeu, ce diagramme de séquence nous permet d'avoir une idée claire sur les informations échangées.

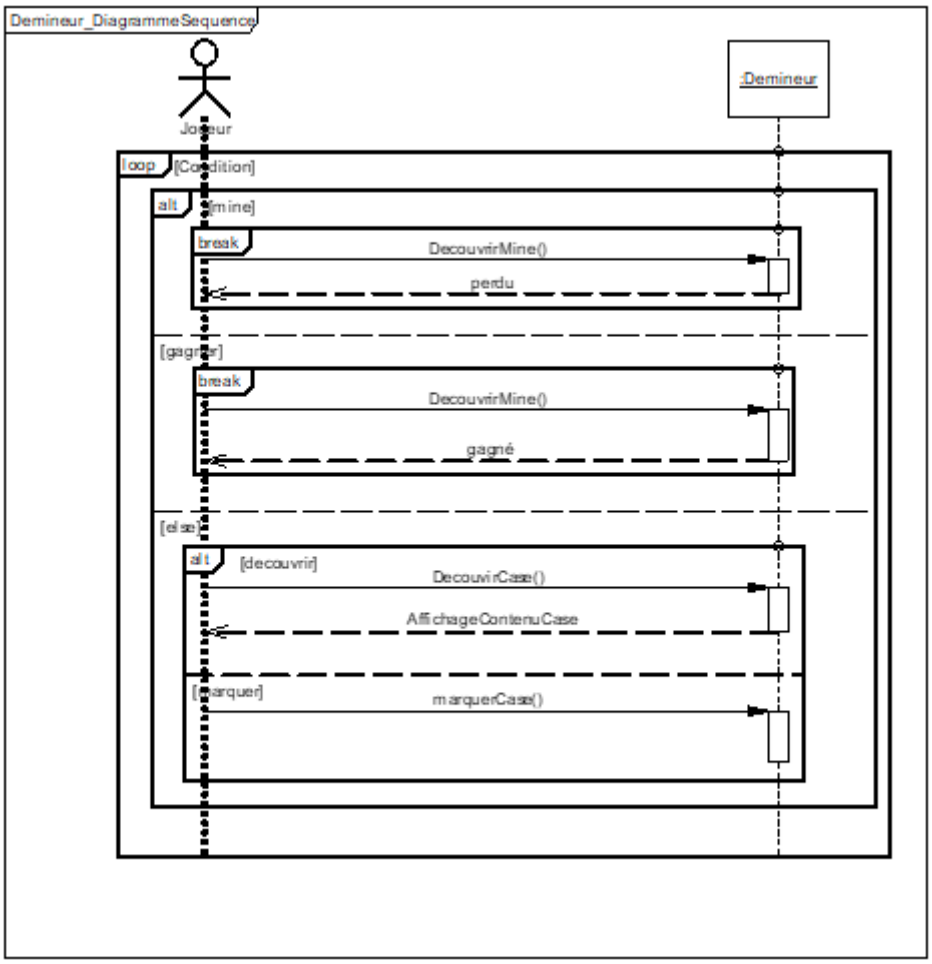


Figure 3 : Diagramme de séquence

2. Diagramme d'activité

Pour appuyer le précédent diagramme et mieux comprendre les interactions entre les éléments de notre jeu nous utiliserons un diagramme d'activité pour expliciter les fonctionnalités et le principe du jeu.

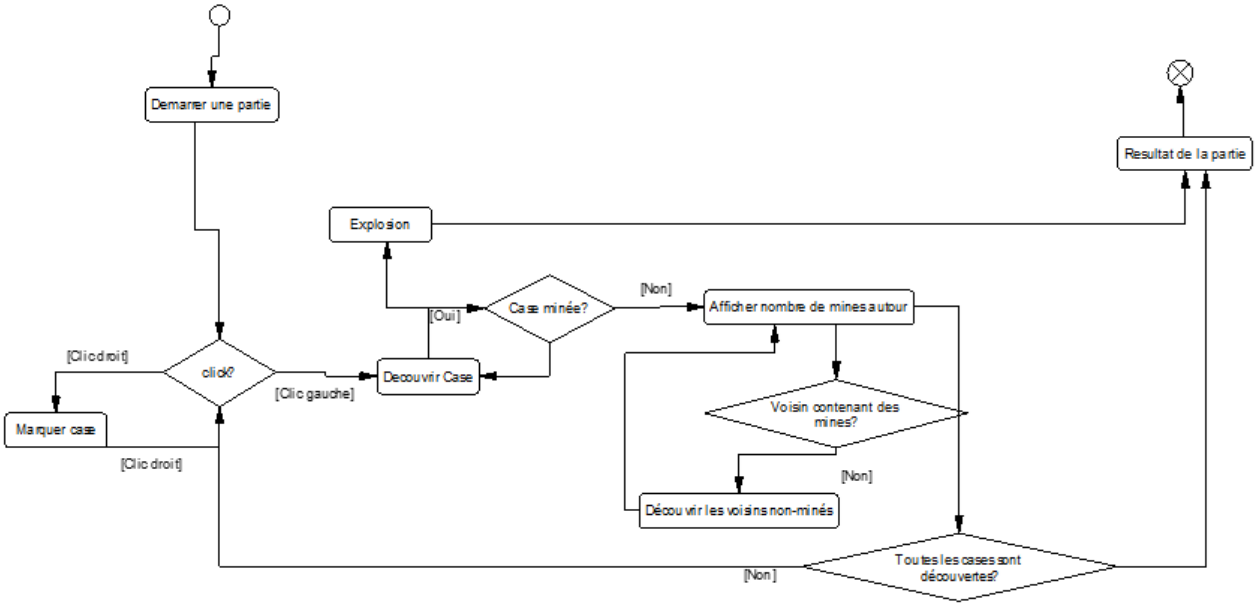


Figure 4 : Diagramme d'activité

3. Diagramme d'état – transition

Au cours du déroulement du jeu l'élément fondamental du jeu change d'état en fonction des actions opérées par le joueur. Le diagramme d'état-transition suivant montre les transformations subies pas une case au cours du jeu.

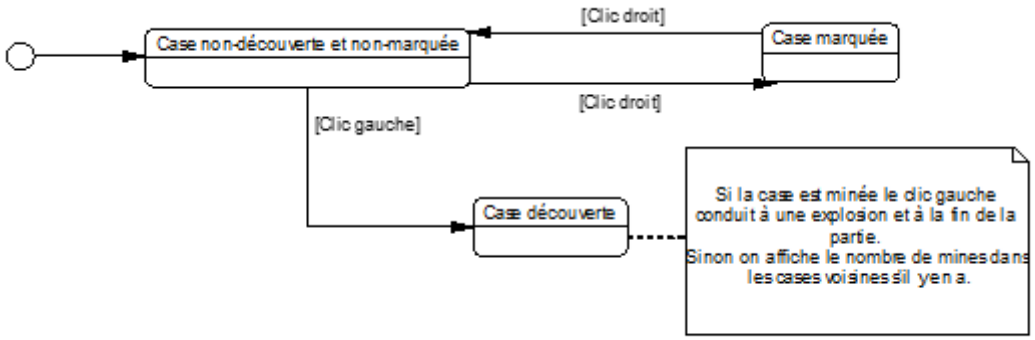


Figure 5: Diagramme d'état-transition d'une case

III. Implémentation

A. Outils de développement : Présentation de C++ et de la bibliothèque Qt

Créé dans les années 1980, le C++ est un langage issu du C dont il inclut tous les éléments en ce qui concerne la conception d'algorithme. Il améliore ce dernier en y ajoutant des fonctionnalités telles que la programmation orientée objet. Le C++ permet donc de développer des logiciels robustes en permettant au développeur d'écrire du code cohérent et respectant les principes fondamentaux d'une bonne programmation.

Pour pouvoir réaliser ce projet nous avons besoin d'un environnement nous permettant d'exploiter au mieux le C++ et toutes ses qualités en termes de performances et de flexibilité. Nous avons donc choisi d'utiliser la bibliothèque **Qt**.

Qt est un framework C++ qui permet de développer des applications en utilisant des composants de conception d'interfaces graphiques(widgets), d'accès aux données, de gestion des communication interprocessus avec les signaux et les slots. Qt est multiplateforme et intègre en son sein un compilateur permettant d'exécuter les mêmes applications sur différentes plateformes. Tous ces éléments sont inclus dans son environnement de développement intégré **QtCreator**.

Nous utiliserons la version 5.12.0 de Qt et la version 4.12.2 de QtCreator

Figure 6 : Logo de Qt

B. Mise en œuvre

1. Principe de développement

Notre jeu est composé de cases qui sont chargées au démarrage de l'application avec la fonction ***DémarrerJeu*** (Accueil.cpp). Cette fonction appelle respectivement ***ConfigMachineEtat***, ***initialisationJeu*** et ***adjustSize***. Ces fonctions sont chargées de configurer et d'initialiser le jeu.

InitialisationJeu charge les cases sur la fenêtre en fonction du niveau choisi. Le jeu démarre par défaut avec le niveau facile. A chaque changement de niveau, la fenêtre est actualisée en prenant en compte les nouveaux paramètres.

Les mines sont réparties de façon aléatoires sur l'ensemble des cases. Des slots sont configurés dans la classe pour récupérer les signaux de victoire ou d'échec lors du déroulement du jeu.

2. Exemple d'implémentation de quelques classes et fonctions

Dans cette section, nous donnerons des détails sur le code de notre application. Nous donnerons brièvement des descriptions sur certaines classes principales, nous expliquerons le principe général de fonctionnement du code au travers des différentes fonctions implémentées

- Classe « Case »

Chaque case du jeu est un QPushButton dont la taille et le style sont préconfigurés. Les différents évènements (clic droit, clic gauche) ont chacun leur slot pour se connecter à une fonction spécifique.

```cpp
#include <QPushButton>
#include <QList>
#include <QStateMachine>
#include <QState>
#include <QFinalState>
#include <QSignalMapper>

class Case : public QPushButton
{
    Q_OBJECT
public:

    Case(Position position, QWidget* parent = nullptr);
    virtual ~Case() override;
    void AfficheResultat();
    void AjoutVoisin(Case* Case);
    Position position() const;
    void placerMine(bool val);

    bool EstMine() const;
    bool AuneMarque() const;
    bool EstOuvert() const;
    bool isUnrevealed() const;
    bool AdesMinesAutour() const;

    unsigned int NbMinesAutour() const;
    unsigned int NbDrapeauAutour() const;

    QList<Case*>& Voisinage();

    virtual void mousePressEvent(QMouseEvent* e) override;
    virtual void mouseReleaseEvent(QMouseEvent* e) override;
    virtual void mouseMoveEvent(QMouseEvent* e) override;
    virtual QSize sizeHint() const override;
    //static QIcon getIcon(IconeCase icone);
```

Figure 7 : Aperçu Case.h

```
void Case::ConfigGestionEtatCase()
{
    ModeNormal = new QState;
    ModeAppercu = new QState;
    AppercuDesVoisins = new QState;
    ModeDrapeauPointe = new QState;
    ModeDecouvert = new QState;
    ModeVoisinsDecouverts = new QState;
    ModeInactive = new QFinalState;
// Gestion des transtions
    // quitter Etat normal de la case aux evenements
    ModeNormal->addTransition(this, &Case::ClickNormal, ModeDecouvert);
    ModeNormal->addTransition(this, &Case::ClickDroit, ModeDrapeauPointe);
    ModeNormal->addTransition(this, &Case::reveal, ModeDecouvert);
    ModeNormal->addTransition(this, &Case::preview, ModeAppercu);
    ModeNormal->addTransition(this, &Case::Inactivation, ModeInactive);

    ModeAppercu->addTransition(this, &Case::reveal, ModeDecouvert);
    ModeAppercu->addTransition(this, &Case::unPreview, ModeNormal);
    ModeAppercu->addTransition(this, &Case::Inactivation, ModeInactive);

    ModeDrapeauPointe->addTransition(this, &Case::ClickDroit, ModeNormal);

    ModeDecouvert->addTransition(this, &Case::ClickGaucheEtDroit, AppercuDesVoisins);

    AppercuDesVoisins->addTransition(this, &Case::PasDeClick, ModeVoisinsDecouverts);
    AppercuDesVoisins->addTransition(this, &Case::unPreview, ModeDecouvert);

    ModeVoisinsDecouverts->addTransition(this, &Case::reveal, ModeDecouvert);

    connect(ModeNormal, &QState::entered, [this]()
    {
        //this->setIcon(getIcon(IconeCase::Vide));
        this->setStyleSheet(NormalStyle);
```

Figure 8 : Aperçu Case.cpp (Configuration des transitions)

- Clase « CompteurDeMines »

Le Compteur de mine décompte en fonction des drapeaux que l'on place au cours du jeu. La fonction SetNombreDrapeau affiche continuellement la différence entre le nombre de mines présentes dans le champ de jeu et le nombre de drapeaux placés par le joueur.

```cpp
#include "CompteurDeMine.h"
#include "Styles.h"

CompteurDeMine::CompteurDeMine(QWidget* parent)
    : QLCDNumber(parent)
{

    this->setSegmentStyle(QLCDNumber::Flat);
    this->display(0);
    this->setStyleSheet(CompteurStyle);
    this->setDigitCount(3);
    this->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
}

void CompteurDeMine::SetNombreMines(int numMines)
{
    nombreTotalMines = numMines;
    display((int)nombreTotalMines);
}

void CompteurDeMine::SetNombreDrapeau(unsigned int flagCount)
{
    nombreDrapeau = flagCount;
    display((int)nombreTotalMines - (int)flagCount);
}

QSize CompteurDeMine::sizeHint() const
{
    return QSize(69, 38);
}
```

Figure 9 : Aperçu CompteurdeMines.cpp

- Classe « Horloge »

Notre classe « Horloge » est chargée d'enregistrer le temps mis par un joueur pour gagner. C'est une sous-classe de QLCDNumber. Elle est exprimée en secondes et s'incrémente automatiquement après le premier clic.

```cpp
#pragma once
#include <QLCDNumber>
#ifndef HORLOGE_H
#define HORLOGE_H
class Horloge: public QLCDNumber
{
public:
    Horloge(QWidget* parent = nullptr);

    void incrementTime();
    int time() const;
    virtual QSize sizeHint() const override;

private:

    int m_seconds;
};
#endif // HORLOGE_H
```

```cpp
#include "Horloge.h"

Horloge::Horloge(QWidget* parent /*= nullptr*/)
    : QLCDNumber(parent)
    , m_seconds(0)
{
    this->setDigitCount(4);
    this->display(0);
    this->setStyleSheet(".QLCDNumber { border: 2px inset gray; }");
    this->setSegmentStyle(QLCDNumber::Flat);
    this->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
}

void Horloge::incrementTime()
{
    display(++m_seconds);
}

int Horloge::time() const
{
    return m_seconds;
}

QSize Horloge::sizeHint() const
{
    return QSize(65, 35);
}
```

- Gestion des niveaux :

Plusieurs niveaux de difficulté sont disponibles dans notre jeu : facile, intermédiaire et expert.

Le mode facile comprend une matrice 10 x 10 cases avec 10 mines. Le mode intermédiaire pour sa part est composé de 256 cases (16 x 16) et 40 mines. Le niveau difficile quant à lui a 99 mines dissimulés dans 480 cases (16 lignes et 30 colonnes). La fonction defNiveau présente dans la classe Accueil (Accueil.cpp) sert à configurer le nombre de lignes, de colonnes et de mines.

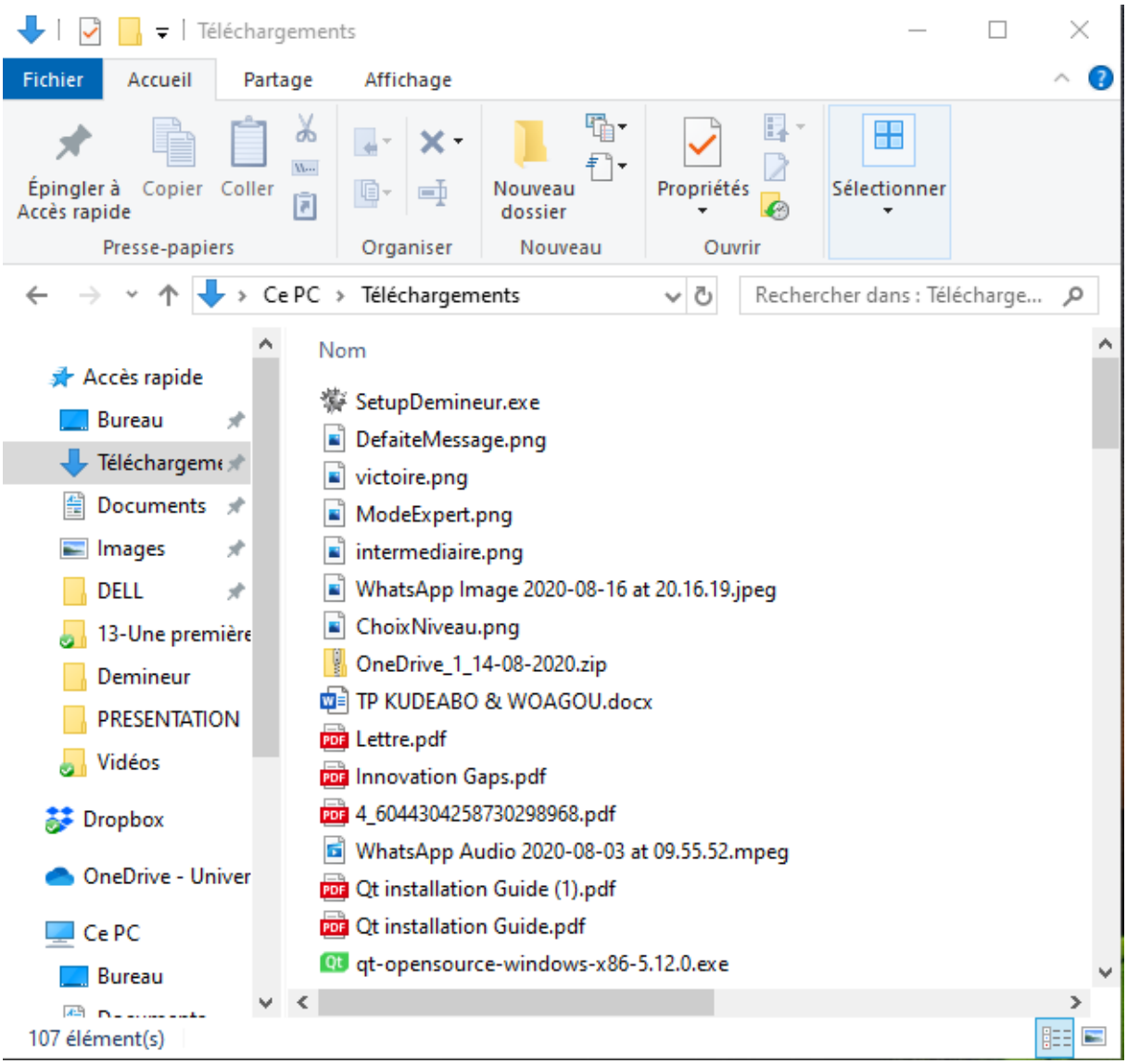
```
void Acceuil::defNiveau(Niveau niveau){
    this->niveau = niveau;

    switch (niveau)
    {
    case Niveau::Facile:
        nbLigne = 10;
        nbCols = 10;
        nbMines = 10;
        facileAction->setChecked(true);
        break;
    case Niveau::Intermediaire:
        nbLigne = 16;
        nbCols = 16;
        nbMines = 40;
        intermediaireAction->setChecked(true);
        break;
    case Niveau::Expert:
        nbLigne = 16;
        nbCols = 30;
        nbMines = 99;
        expertAction->setChecked(true);
        break;
    default:
        break;
    }
    DemarrerNouveuJeu();
}
```

IV. Mode d'utilisation

A. Guide d'installation

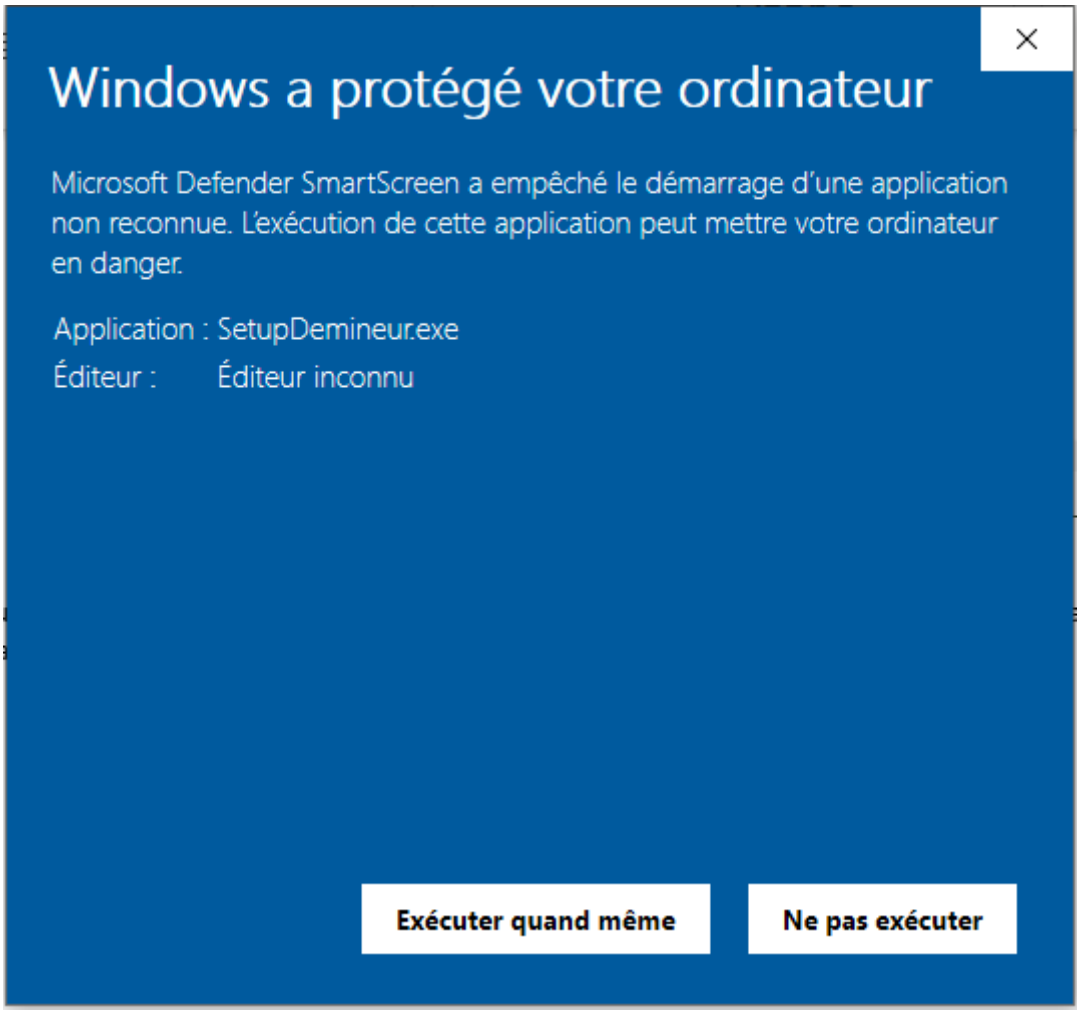
Dans cette partie nous donnerons des directives par rapport à l'installation de notre appliction.

1. Double-cliquer sur l'exécutable « SetupDemineur.exe »

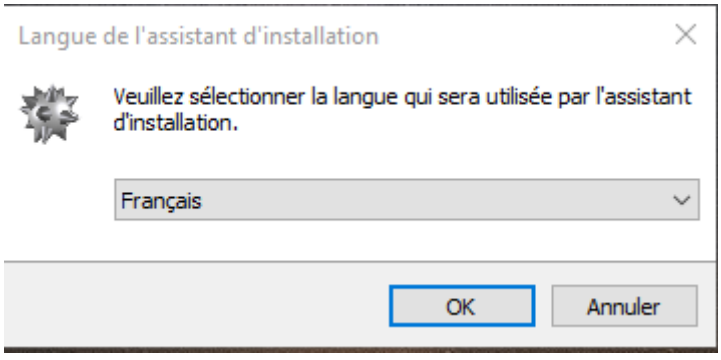


2. Contrôle de sécurité

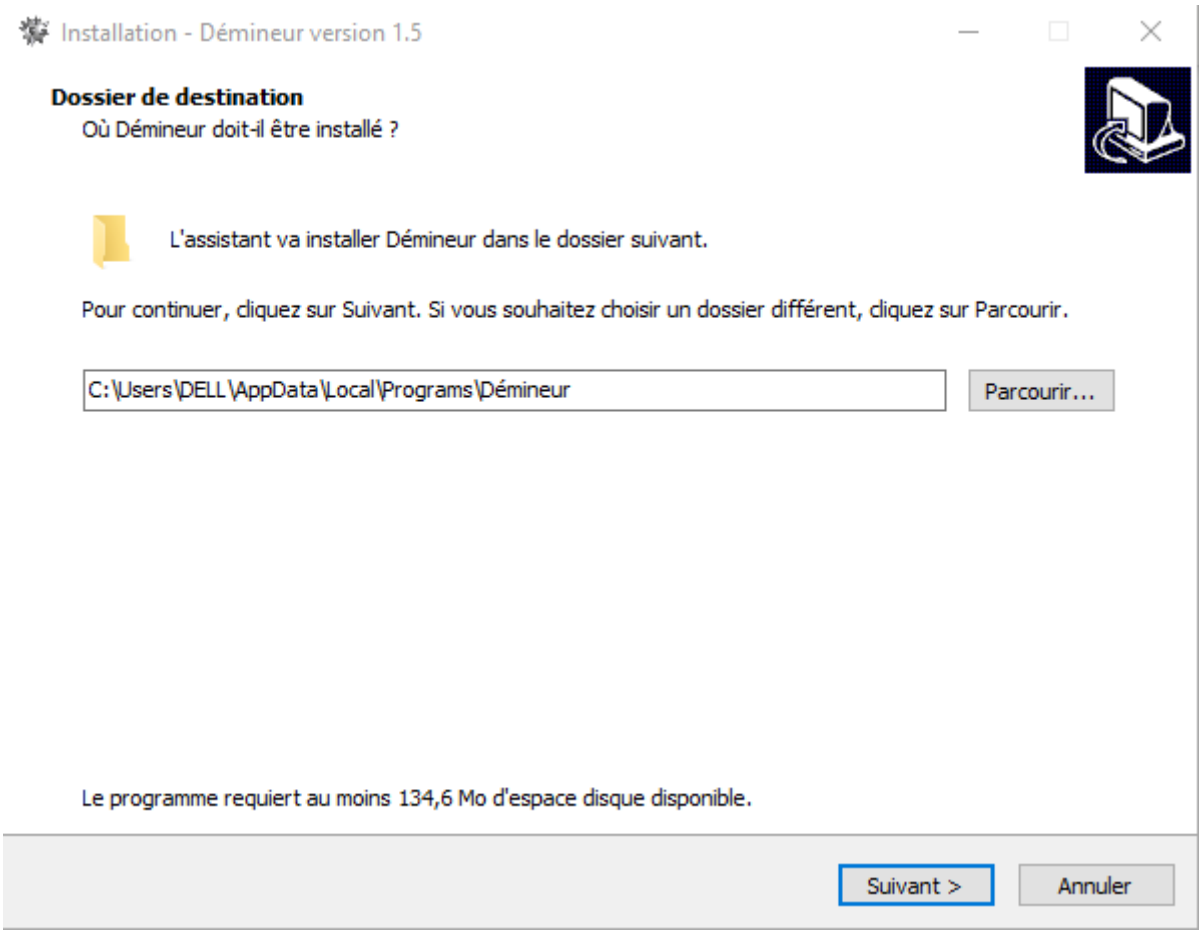
Dans le cas où votre système d'exploitation vous déconseille de l'installer vous pouvez passez ce contrôle en cliquant sur « Exécuter quand même » (Exemple de Windows)



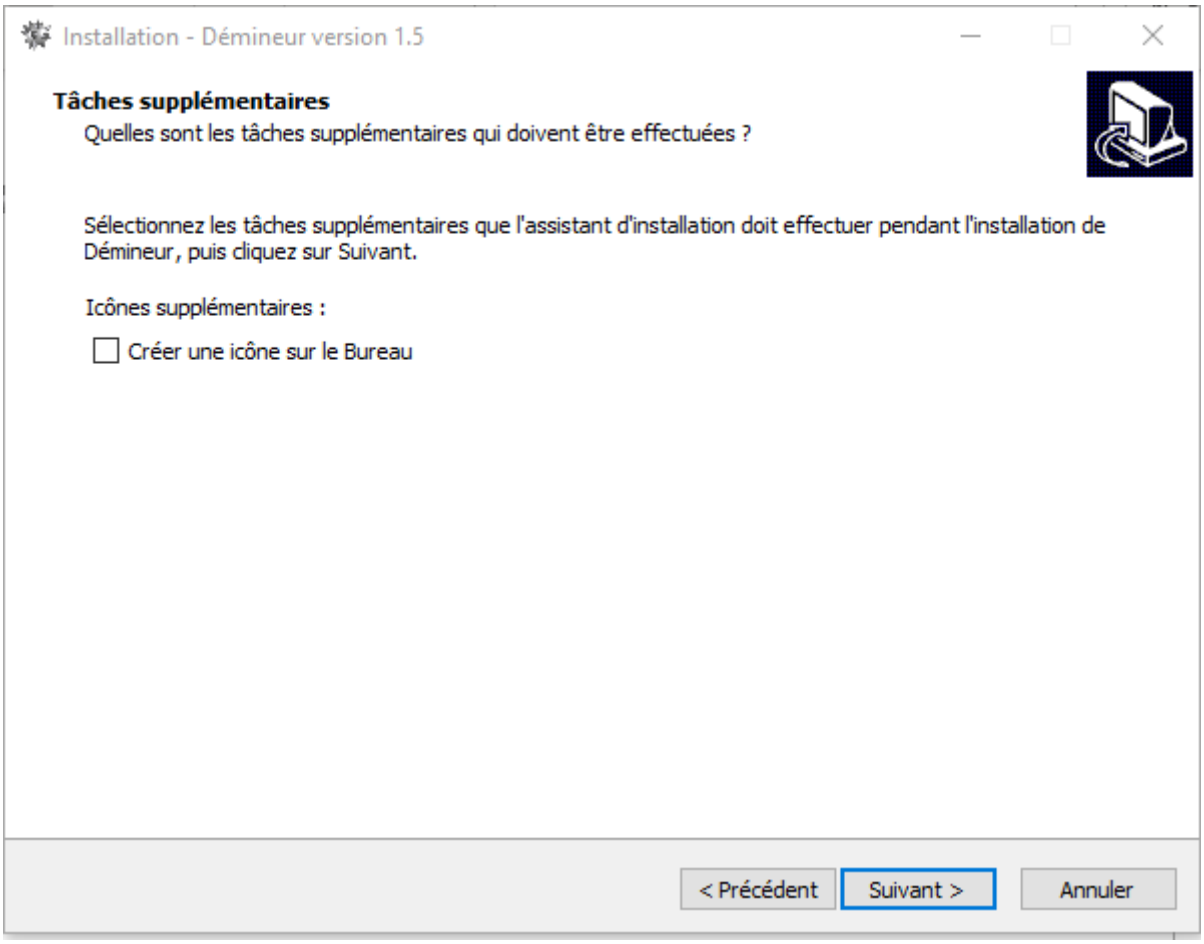
3. Choix de la langue



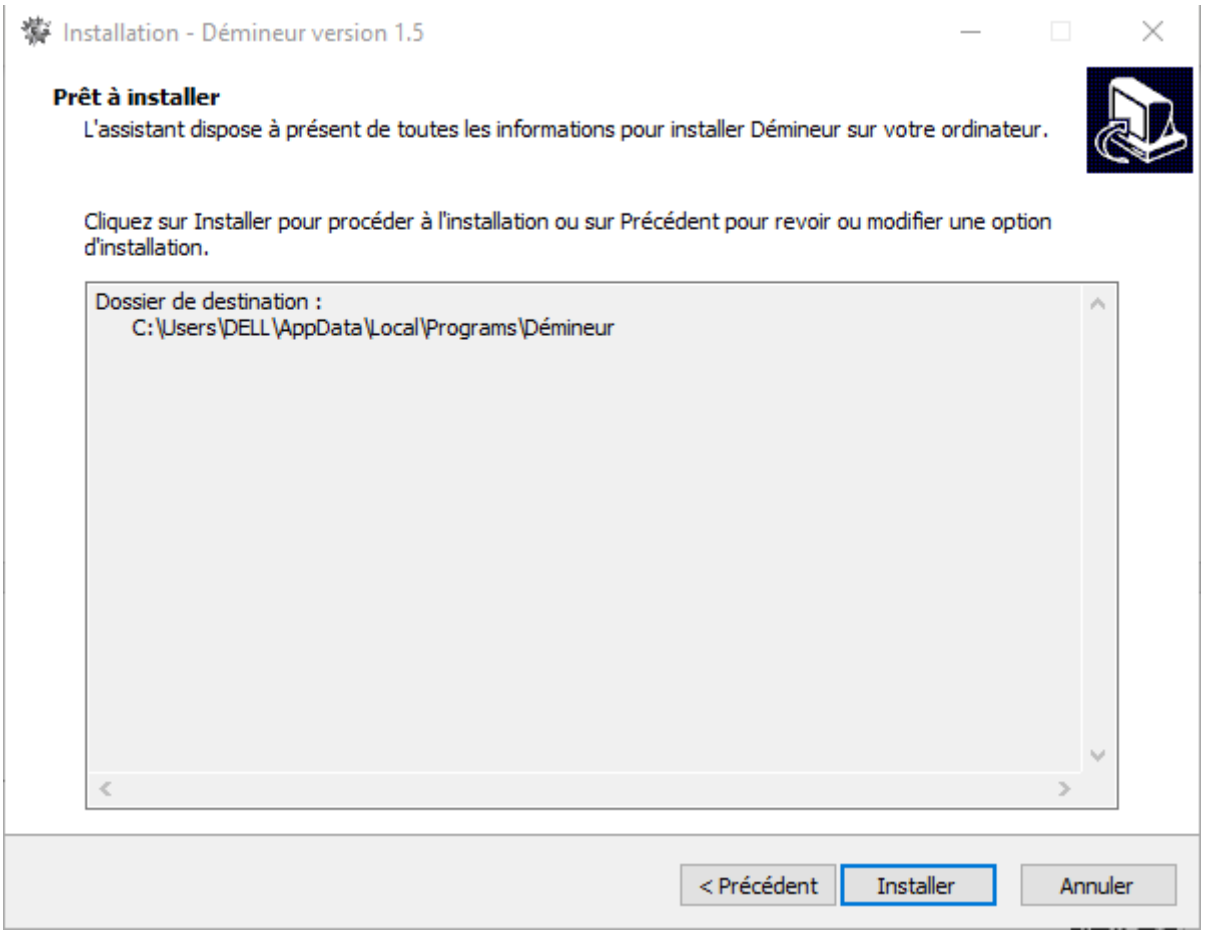
4. Choix du dossier d'installation



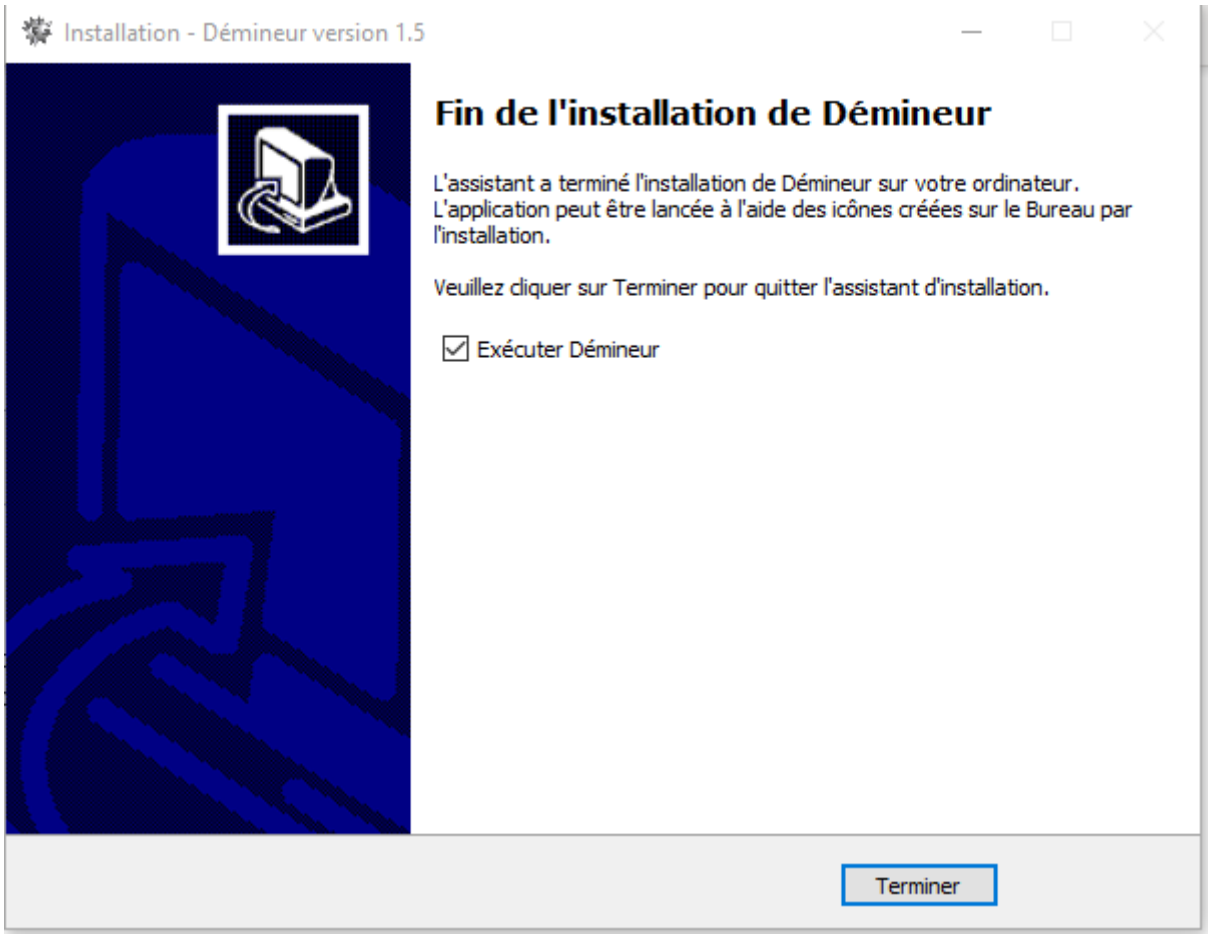
5. Choix d’avoir une icône sur le bureau ou non



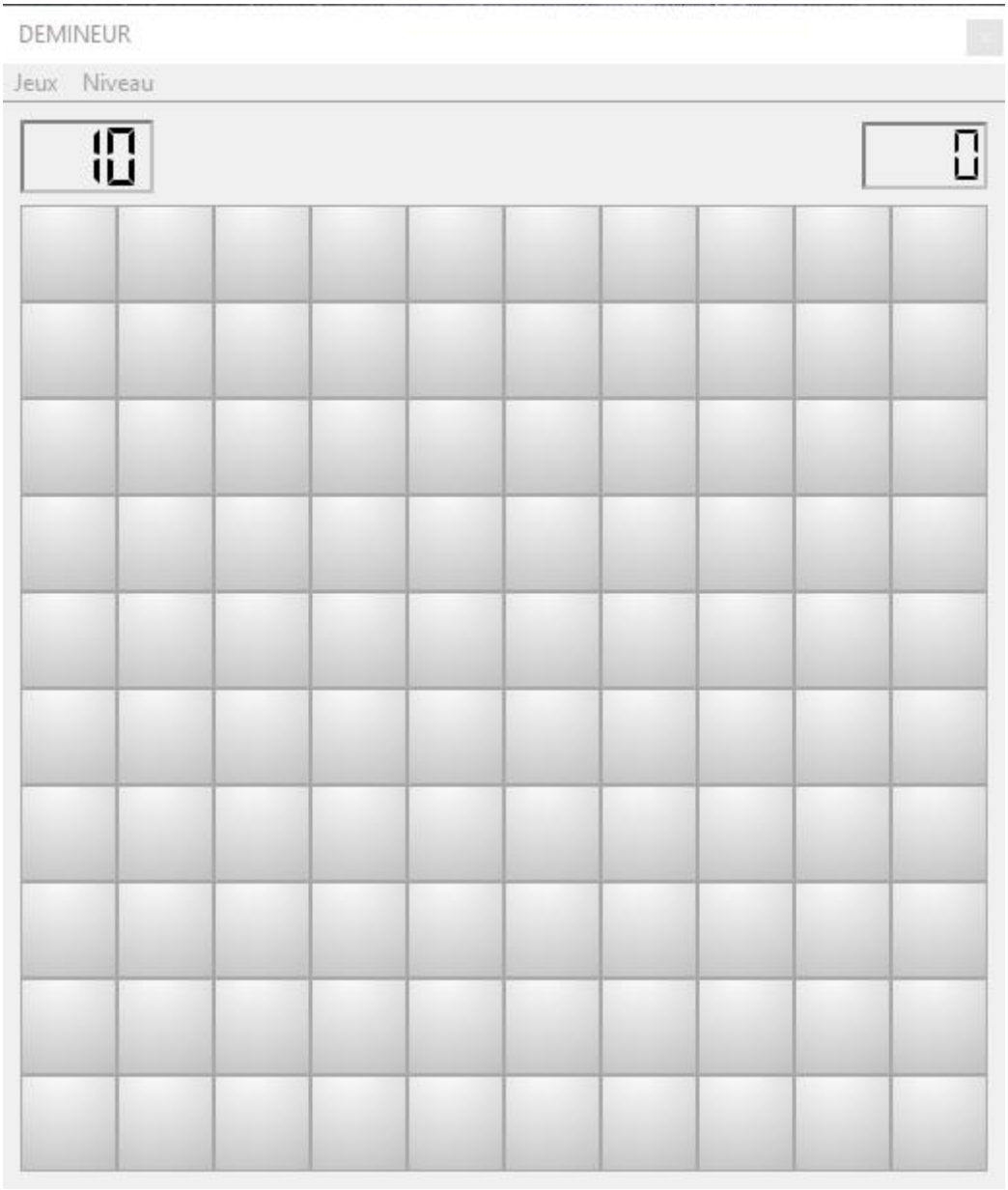
6. Confirmation des différents paramètres et lancement de l'installation



7. Fin de l’installation : vous pouvez choisir d’exécuter le jeu en même temps



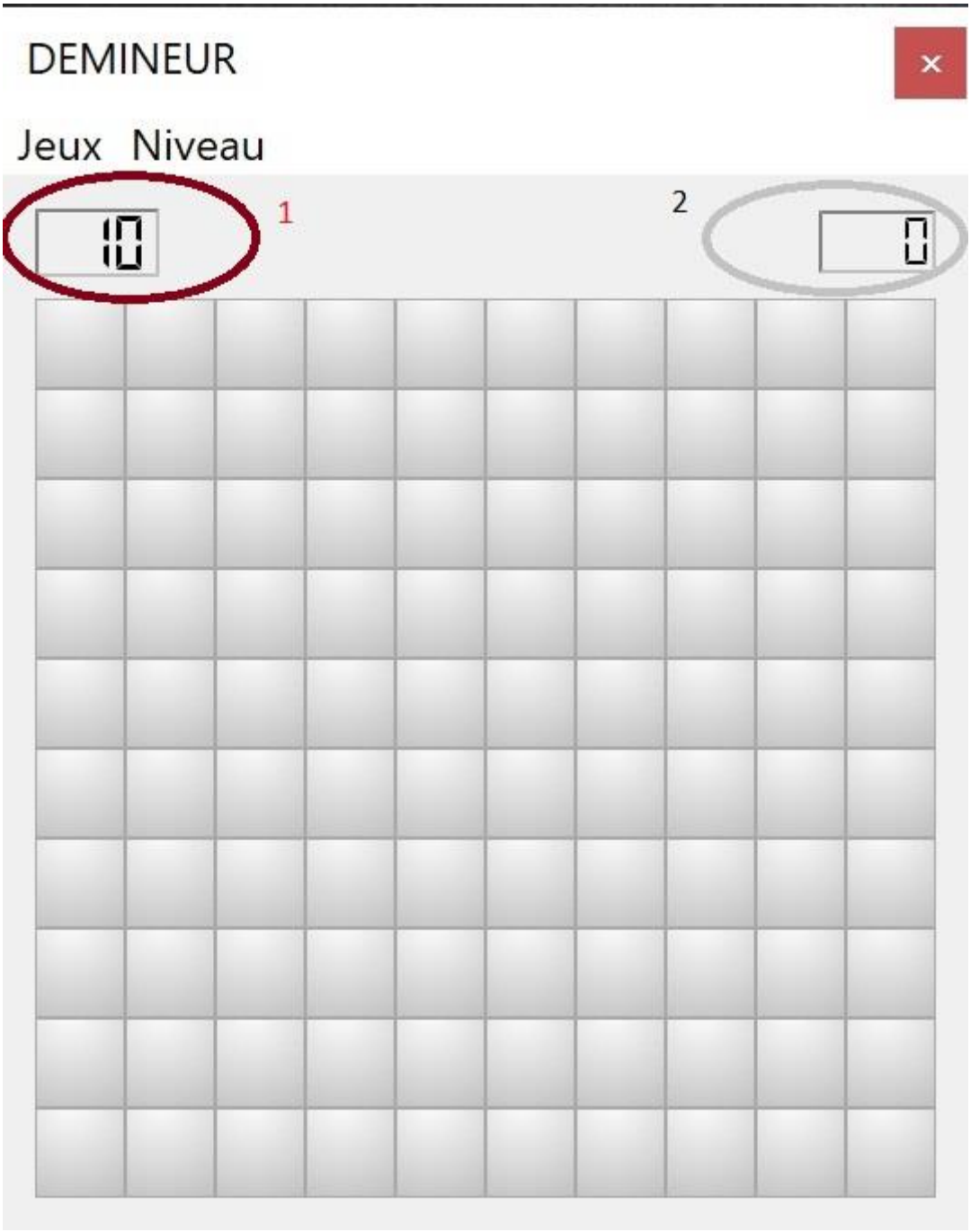
8. Lancement du jeu



B. Présentation de l'application

Après un double-clic pour lancer notre application, la fenêtre qui nous est présentée est la suivante. Nous rappelons que le jeu démarre par défaut avec le niveau facile.

- Accueil

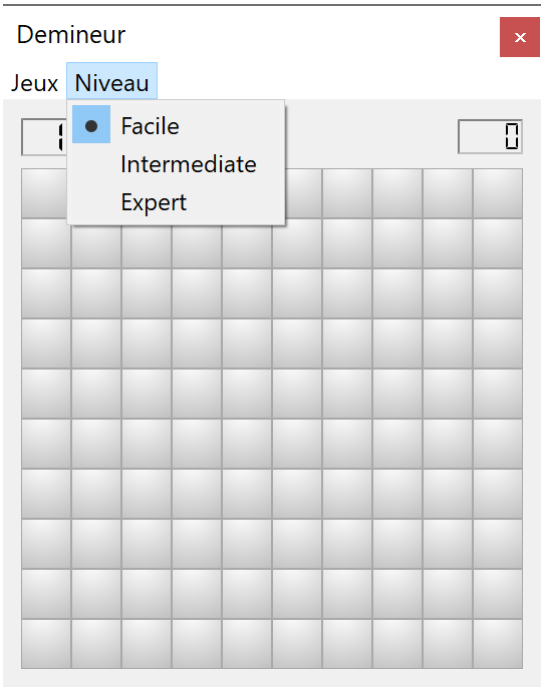


Légende :

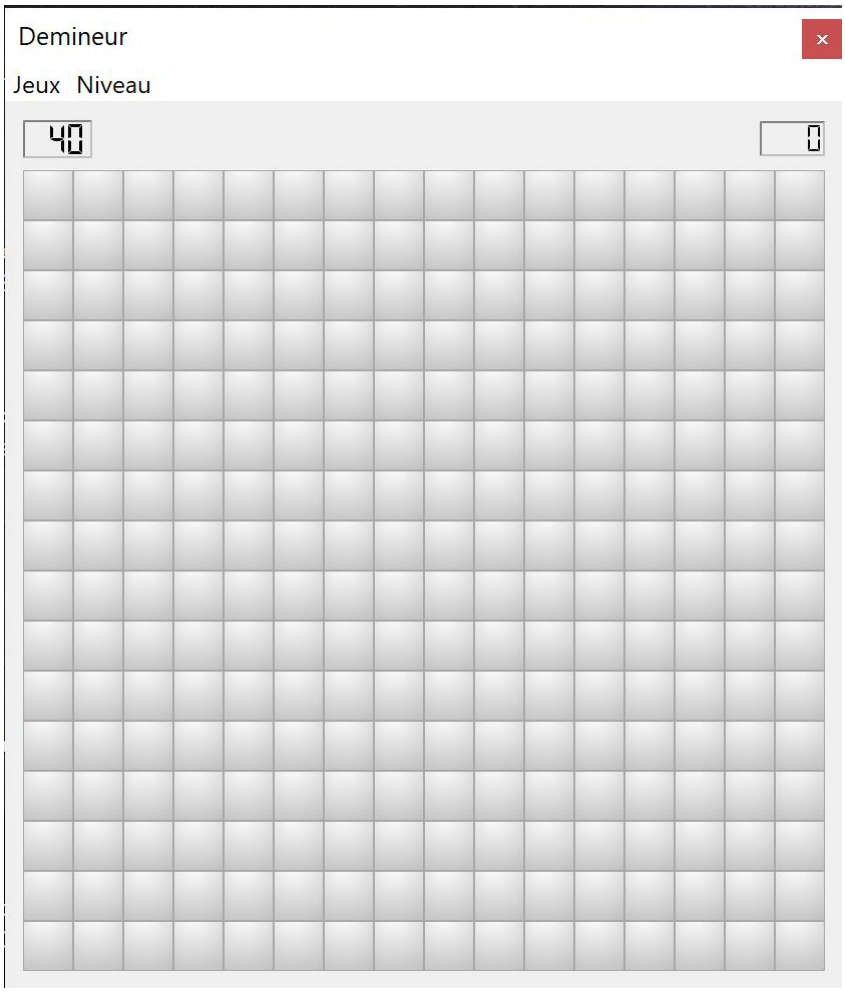
1. Nombre de mines
2. Chronomètre

- Niveaux

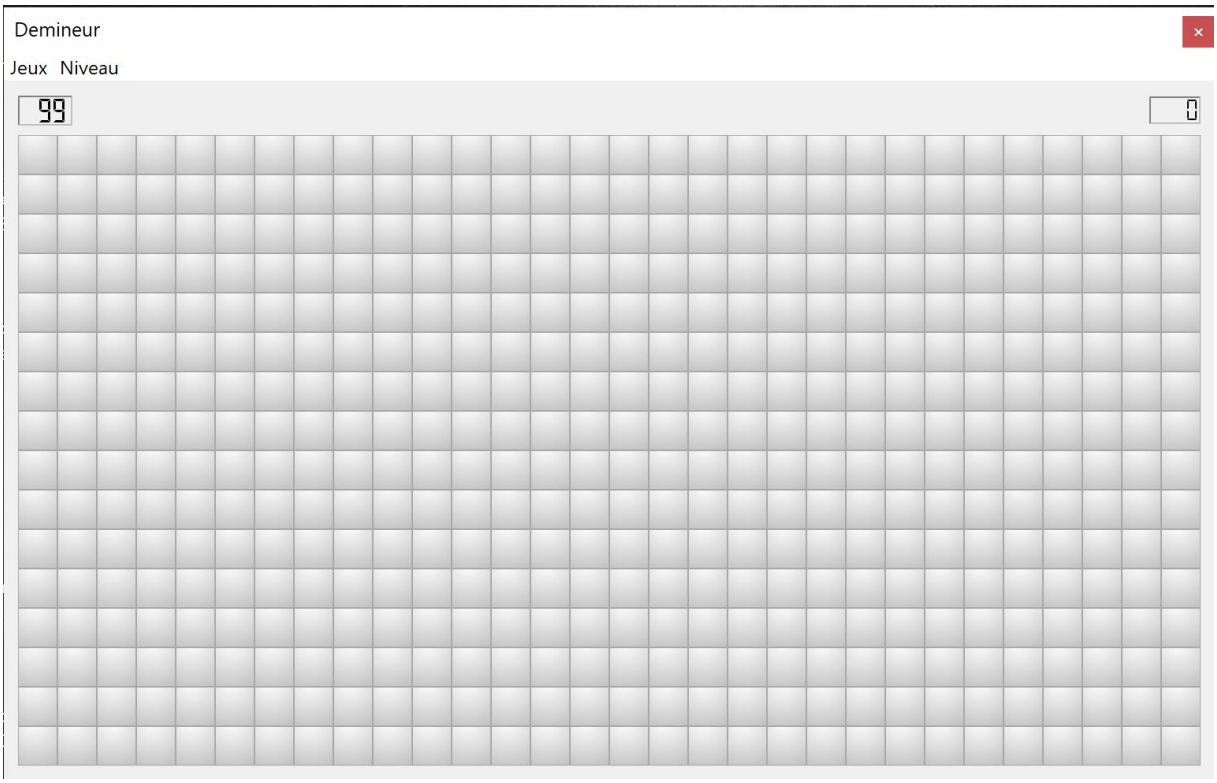
En cliquant sur le sous-menu « Niveau », on a la possibilité de choisir le niveau de difficulté. Après le choix du niveau, la fenêtre se recharge en présentant le nombre de cases correspondant au niveau choisi.



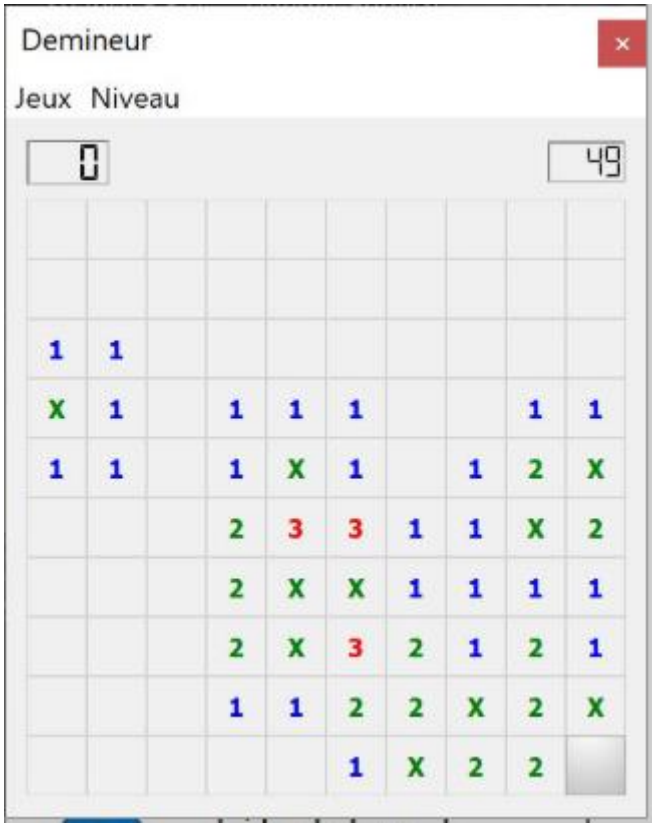
- Niveau « intermédiaire »



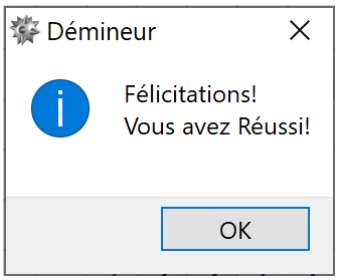
- Niveau « expert »



- Exemple fin de jeu (toutes les mines sont découvertes)



- Notification de victoire



- Notification de défaite après clic sur une mine

- Nouveau jeu (Jeux)

Un clic sur le bouton jeu permet « Jeux » permet de recommencer une nouvelle partie en gardant le même niveau que la partie actuelle

Conclusion

Ce cours de programmation orientée objet avancée que nous avons eu à suivre, nous a permis d'approfondir nos connaissances en matière de conception objet et de nous familiariser avec le C++ qui est un langage conçu pour développer des logiciels basés sur des architectures objet. Ce projet nous a aidé d'associer les principes appris durant le cours à la programmation en C++ avec le framework Qt.

Table des matières