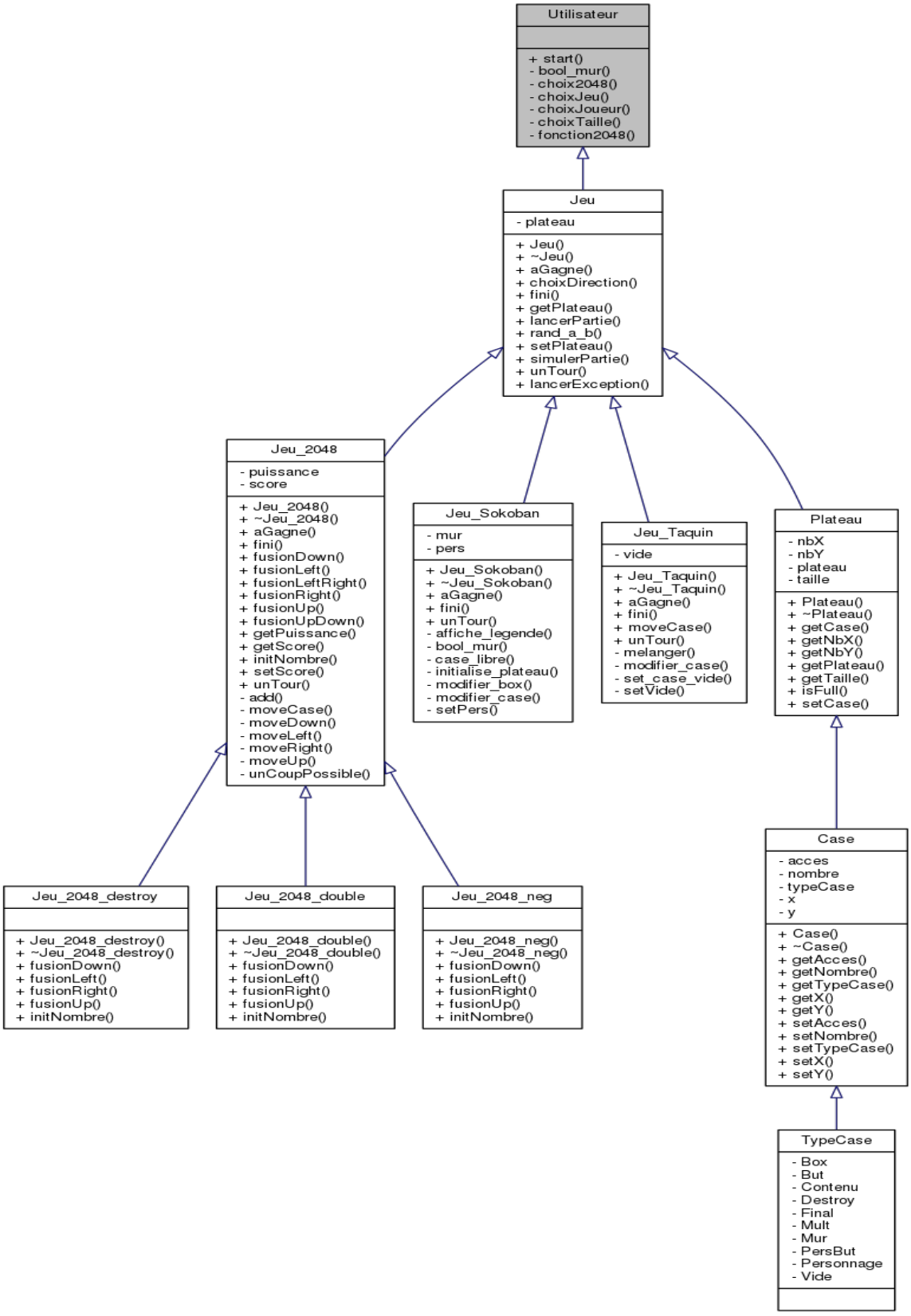
Programmation Orienté Objet

Modélisation du Projet

C++

BESSA Alexandre & JACQUETTE Pierrick
21306128 & 21305551

Diagramme de classes



Modélisation

enum class TypeCase

```
{
   Contenu, Vide, Mult, Destroy, Mur, But, Personnage, Box, Final, PersBut
};
std::ostream& operator<<(std::ostream&, const TypeCase);
```

Cette classe représente les différents types possible pour une case du plateau. Mur, But et Personnage serviront pour le Sokoban.

class Case

```
{
private :
        TypeCase typeCase;
        int nombre; //valeur de la case
        int x;
        int y;
        bool acces; //pour le 2048 permet de savoir si la case est deja modifie à chaque tour
public :
        Case(TypeCase=TypeCase::Vide);
        virtual ~Case();
        int getNombre() const;
        bool getAcces() const;
        TypeCase getTypeCase() const;
        void setNombre(int);
        void setAcces(bool);
        void setTypeCase(TypeCase);
        void setX(int x);
        void setY(int y);
        int getX();
        int getY();
        friend std::ostream & operator<<(std::ostream&, const Case);
};
```

Cette classe représente l'objet case, le nombre permet de représenter la valeur d'une case. Le type de la case peut varier dans le temps.

class Plateau

```
{
private :
        int taille;
        int nbX;        int nbY;
        std::vector<std::vector<Case>> plateau;
public :
        Plateau(int x, int y);
        virtual ~Plateau();
        int getTaille() const;
        int getNbX() const;
        int getNbY() const;
        std::vector<std::vector<Case>> getPlateau() const;
        Case getCase(int, int);
        void setCase(Case, int, int);
        bool isFull() const;
        friend std::ostream &operator <<(std::ostream &, const Plateau) ;
};
```

Cette classe représente l'objet Plateau, on la stocke dans un tableau (vector).

class Jeu

```
{
private :
        Plateau plateau;
public :
        Jeu(int x, int y);
        virtual ~Jeu();
        Plateau getPlateau() const;
        void setPlateau(Plateau);
        int rand_a_b(int, int);
        int choixDirection();
        virtual bool fini()=0; // test si le jeu est fini
        virtual void unTour(int)=0;
        virtual bool aGagne()=0; // Test si on a gagné
        void lancerPartie();
        void simulerPartie();
        static bool lancerException(bool, std::string);
};
```

Cette classe est une classe abstraite, les méthodes seront redéfinies dans les classes dérivées. Cela permettra d'adapter les règles, l'affichage… en fonction de chaque jeu que l'on souhaite implémenter. On peut faire la différences entre une partie pour un joueur humain et celle pour un robot.

```
class Jeu_Taquin : public Jeu
{
private :
        Case vide;
        void setVide(int i, int j);
        void modifier_case(Plateau p, int i, int j);
        void melanger(int*, int );
        void set_case_vide(Plateau p, int x, int y);
public :
        Jeu_Taquin(int x, int y );
        ~Jeu_Taquin();
        bool fini();
        void unTour(int);
        bool aGagne();
        bool moveCase(int);
};
```

Cette classe représente le jeu du Taquin, elle hérite de la classe Jeu en redéfinissant ces méthodes.

```
class Jeu_Sokoban : public Jeu
{
private :
        Case pers;
        bool mur;
        void initialise_plateau(TypeCase* , int);
        bool case_libre(Plateau p, int i, int j);
        void setPers(Plateau p, int i, int j);
        void modifier_case(Plateau p, int i, int j);
        void modifier_box(Plateau p, int i, int j, int iprec, int jprec);
        bool bool_mur();
        void affiche_legende();
public :
        Jeu_Sokoban(int x, int y, bool boolean);
        virtual ~Jeu_Sokoban();
        bool fini();
```

```
        void unTour(int);
        bool aGagne();
};
```

Cette classe représente le jeu du Sokoban, elle hérite de la classe Jeu en redéfinissant ces méthodes.

```
class Jeu_2048 : public Jeu
{
private :
        int score;
        int puissance;
        void add();
        void moveDown();
        void moveUp();
        void moveLeft();
        void moveRight();
        bool unCoupPossible();
        void moveCase(int);
public :
        Jeu_2048(int x, int y, int puissance=2);
        int getScore() const;
        int getPuissance() const;
        void setScore(int);
        virtual ~Jeu_2048();
        bool fini();
        void unTour(int);
        bool aGagne();
        virtual Case initNombre(Case);
        virtual void fusionLeft(int, int);
        virtual void fusionRight(int, int);
        virtual void fusionUp(int, int);
        virtual void fusionDown(int, int);
        void fusionLeftRight(int, int, int, bool);
        void fusionUpDown(int, int, int, bool);
};
```

Cette classe représente le jeu du 2048, elle hérite de la classe Jeu en redéfinissant ces méthodes. Ceci est le jeu de base.

```
class Jeu_2048_destroy : public Jeu_2048
{
public:
	Jeu_2048_destroy(int x, int y);
	~Jeu_2048_destroy();
	Case initNombre(Case);
	void fusionLeft(int, int);
	void fusionRight(int, int);
	void fusionUp(int, int);
	void fusionDown(int, int);
};
```

Cette classe représente le jeu du 2048 , elle hérite de la classe Jeu en redéfinissant ces méthodes. Ceci est le jeu 2048 avec la fonctionnalité de case destroy qui permettra de supprimer la tuile qui fusionne avec elle.

```
class Jeu_2048_neg : public Jeu_2048
{
public:
	Jeu_2048_neg(int x, int y);
	~Jeu_2048_neg();
	Case initNombre(Case);
	void fusionLeft(int, int);
	void fusionRight(int, int);
	void fusionUp(int, int);
	void fusionDown(int, int);
};
```

Cette classe représente le jeu du 2048 , elle hérite de la classe Jeu en redéfinissant ces méthodes. Ceci est le jeu 2048 avec la fonctionnalité de case négative qui permettra de supprimer un nombre opposé.

```
class Jeu_2048_double : public Jeu_2048
{
public:
	Jeu_2048_double(int x, int y);
	~Jeu_2048_double();
	Case initNombre(Case);
	void fusionLeft(int, int);
	void fusionRight(int, int);
	void fusionUp(int, int);
	void fusionDown(int, int);
};
```

Cette classe représente le jeu du 2048 , elle hérite de la classe Jeu en redéfinissant ces méthodes. Ceci est le jeu 2048 avec la fonctionnalité de case double qui permettra de multiplier par deux la valeur d'une case.

class Utilisateur

```
{
public:
	static void start();
private:
	static int choixTaille(std::string);
	static int choixJeu();
	static int choix2048();
	static int choixJoueur();
	static Jeu* fonction2048(int, int, int);
	static bool bool_mur();
};
```

Cette classe permet de récupérer les choix de l'utilisateur et de lancer le jeu par la suite.