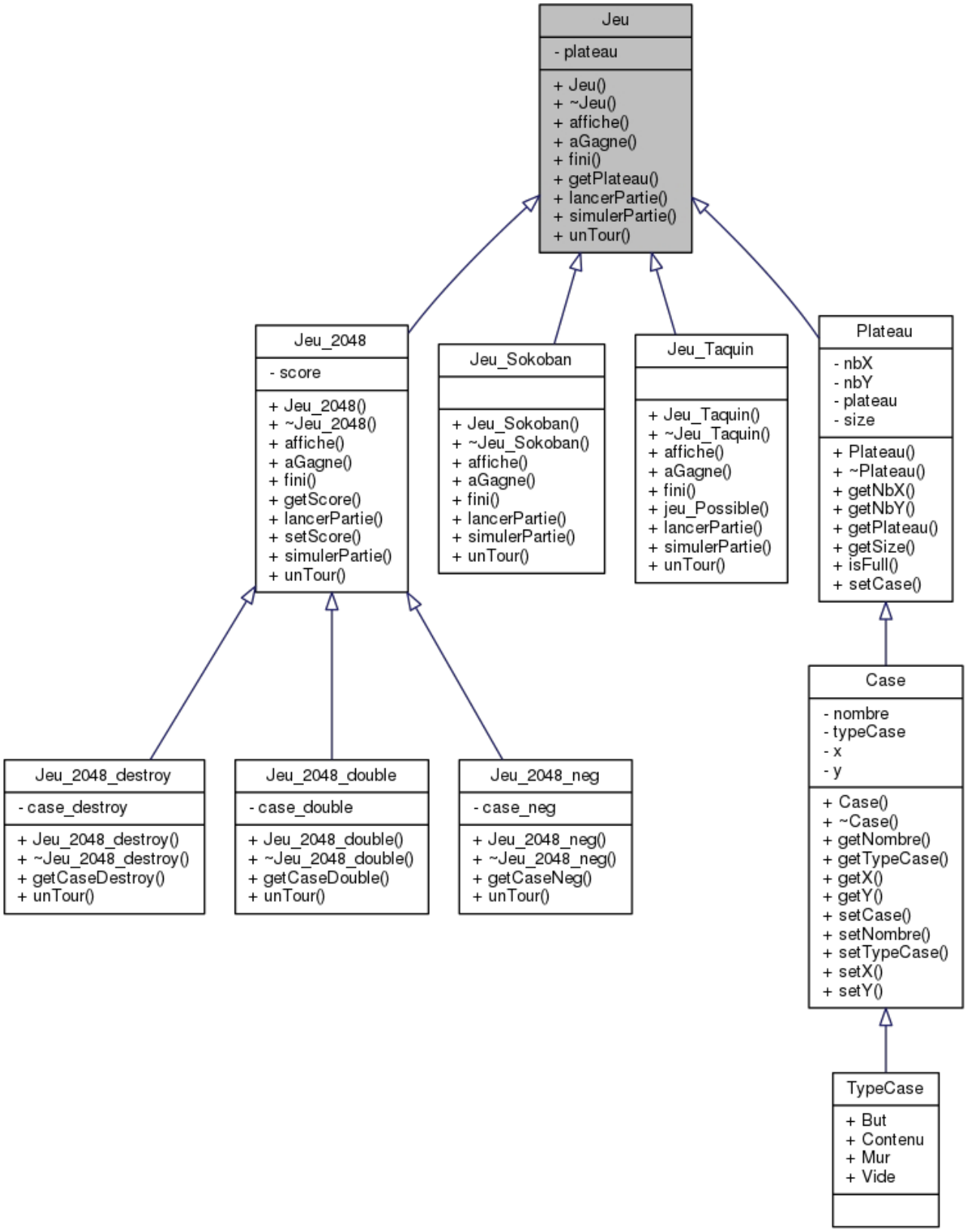
Programmation Orienté Objet

Modélisation du Projet

C++

BESSA Alexandre & JACQUETTE Pierrick
21306128 & 21305551

Diagramme de classes



Modélisation

```
enum class TypeCase{
    Contenu, Vide, Mur, But, Personnage
 };
// Contenu = 0, Vide =1, Mur=2, But=3, Personnage=4
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; //type de la case courante
    int x,
    int y;
    int nombre; //valeur de la case courante
public :
    Case(TypeCase, int, int);
    virtual ~Case();
    int getX() const;
    int getY() const;
    int getNombre() const;
    TypeCase getTypeCase() const;
    void setX(int);
    void setY(int);
    void setNombre(int);
    void setTypeCase(TypeCase);
```

```
    void setCase(TypeCase, int, int);
    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 size;
    int nbX; //dimension du plateau en largeur
    int nbY;  //dimension du plateau en hauteur
    std::vector <Case> plateau; //tableau de case
public :
    Plateau(int x, int y);
    virtual ~Plateau();
    int getSize() const; // retourne x*y
    int getNbX() const;
    int getNbY() const;
    std::vector<Case> getPlateau() const;
    void setCase(Case c);
    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;
    virtual bool fini()=0; // test si le jeu est fini
    virtual void affiche()=0; // affiche en mode texte le jeu
     virtual void unTour()=0; // fait un tour, ici seront lancé les action bougé a droite, gauche... Puis les test sur la possibilité du coup et peut être a diviser en 2
    virtual void aGagne()=0; // Test si on a gagné
    virtual void lancerPartie()=0; // lance une partie pour un humain
    virtual void simulerPartie()=0; // lance une partie pour le robot
    //virtual void newPartie(); Pour faire la difference entre un joueur et un robot
    //virtual void annuleUnTour();
};
```

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 {
public :
    Jeu_Taquin(int x, int y );
    virtual ~Jeu_Taquin();
    virtual bool fini();
    virtual void affiche();
```

```
    virtual void unTour();
    virtual void aGagne();
    virtual void lancerPartie();
    virtual void simulerPartie();
    virtual bool jeu_Possible(); // test si le jeu est possible
};
```

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

```
class Jeu_Sokoban : public Jeu {
public :
    Jeu_Sokoban(int x, int y);
    virtual ~Jeu_Sokoban();
    virtual bool fini();
    virtual void affiche();
    virtual void unTour();
    virtual void aGagne();
    virtual void lancerPartie();
    virtual void simulerPartie();
};
```

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;
public :
    Jeu_2048(int x, int y);
```

```
    int getScore() const;
    void setScore();
    virtual ~Jeu_2048();
    virtual bool fini();
    virtual void affiche();
    virtual void unTour();
    virtual void aGagne();
    virtual void lancerPartie();
    virtual void simulerPartie();
};
```

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);
    virtual ~Jeu_2048_destroy();
    bool getCaseDestroy() const;
    virtual void unTour();
private:
    bool case_destroy;
};
```

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);

bool getCaseNeg() const;

virtual ~Jeu_2048_neg();

virtual void unTour();

private:

bool case_neg;

};

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);

virtual ~Jeu_2048_double();

bool getCaseDouble() const;

virtual void unTour();

private:

bool case_double;

};

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.