Projet LO21 - UTC - P22

Projet de l'UV LO21 réalisé par DU Sylvain, BOUZAR Massil, OUEDRAOGO Taoufiq,
THIBAULT Ambroise et TAVERNY Kelyan

Chapitre 1

Projet Carcassonne - Conception et développement : d'une application pour jouer au jeu Carcassonne

Projet de l'UV LO21 réalisé par DU Sylvain, BOUZAR Massil, OUEDRAOGO Taoufiq, THIBAULT Ambroise et TAVERNY Kelyan

Chapitre 2

Index des espaces de nommage

2.1 Liste des espaces de nommage

Liste de tous les espaces de nommage documentés avec une brève description:

Chapitre 3

Index hiérarchique

3.1 Hiérarchie des classes

Cette liste d'héritage est classée approximativement par ordre alphabétique :

Chapitre 4

Index des classes

4.1 Liste des classes

Liste des classes, structures, unions et interfaces avec une brève description :

Chapitre 5

Index des fichiers

5.1 Liste des fichiers

Liste de tous les fichiers documentés avec une brève description :

Chapitre 6

Documentation des espaces de nommage

6.1 Référence de l'espace de nommage std

6.1.1 Description détaillée

Pour le placement des tuiles, seul l'extension rivière est différente Pour le placement des meeples: -Standard: on peut placer partout sauf sur dans les champs (faite)

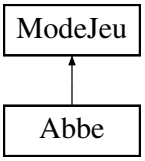
Pour le placement des meeples: -Standard: on peut placer partout sauf sur dans les champs (faite) -Auberge et cathédrales : on ne peut pas placer sur une auberge ou une cathédrale (faite) -Riviere : On ne peut pas placer de meeple sur une tuile de type rivière (faite) -Paysan : couche (récuperer que à la fin de la partie) ou pas couche -Abbe :L'abbé ne peut être posé que sur une abbaye ou un jardin, -tandis que le meeple selon les règles habituelles peut être posé sur n'importe quelle section de tuile (mais pas sur le jardin).

Chapitre 7

Documentation des classes

7.1 Référence de la classe Abbe

Graphe d'héritage de Abbe:



Fonctions membres publiques

- bool validationPlacementT (Tuile newTuile, int x, int y, Plateau ∗plateau)
  Vérifie le placement d'une tuile selon les règles de l'extensions.
- bool validationPlacementM (const TypesTuiles &tp, Meeple ∗m, Espace ∗e)
  Vérifie le placement d'un meeple selon les règles de l'extension.
- void affichage ()

7.1.1 Documentation des fonctions membres

7.1.1.1 affichage()

```
void Abbe::affichage ( )  [inline], [virtual]
```

Implémente ModeJeu.

7.1.1.2 validationPlacementM()

```
bool Abbe::validationPlacementM (
            const TypesTuiles & tp,
            Meeple * m,
            Espace * e )  [inline], [virtual]
```

Vérifie le placement d'un meeple selon les règles de l'extension.

Paramètres

tp	
m	
e	

Renvoie

Implémente ModeJeu.

7.1.1.3 validationPlacementT()

```
bool Abbe::validationPlacementT (
            Tuile newTuile,
            int x,
            int y,
            Plateau * plateau )  [inline], [virtual]
```

Vérifie le placement d'une tuile selon les règles de l'extensions.

Paramètres

newTuile	
x	
y	
plateau	

Renvoie

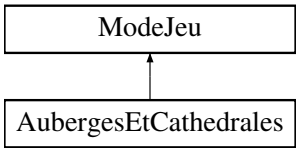
Implémente ModeJeu.

La documentation de cette classe a été générée à partir du fichier suivant :

— modeJeu.h

7.2 Référence de la classe AubergesEtCathedrales

Graphe d'héritage de AubergesEtCathedrales:



Fonctions membres publiques

- bool validationPlacementT (Tuile newTuile, int x, int y, Plateau ∗plateau)
  Vérifie le placement d'une tuile selon les règles de l'extensions.
- bool validationPlacementM (const TypesTuiles &tp, Meeple ∗m, Espace ∗e)
  Vérifie le placement d'un meeple selon les règles de l'extension.
- void affichage ()

7.2.1 Documentation des fonctions membres

7.2.1.1 affichage()

```
void AubergesEtCathedrales::affichage ( )  [inline], [virtual]
```

Implémente ModeJeu.

7.2.1.2 validationPlacementM()

```
bool AubergesEtCathedrales::validationPlacementM (
            const TypesTuiles & tp,
            Meeple * m,
            Espace * e )  [inline], [virtual]
```

Vérifie le placement d'un meeple selon les règles de l'extension.

Paramètres

tp	
m	
e	

Renvoie

Implémente ModeJeu.

7.2.1.3 validationPlacementT()

```
bool AubergesEtCathedrales::validationPlacementT (
            Tuile newTuile,
            int x,
            int y,
            Plateau * plateau )  [inline], [virtual]
```

Vérifie le placement d'une tuile selon les règles de l'extensions.

Paramètres

newTuile	
x	
y	
plateau	

Renvoie

Implémente ModeJeu.

La documentation de cette classe a été générée à partir du fichier suivant :
— modeJeu.h

7.3 Référence de la classe ContenanceTuile

Fonctions membres publiques

— ContenanceTuile (const TypesTuiles t, const int n)
il prend en paramètre nbT.
— const TypesTuiles & **getType** () const
— const int **getNumPlacement** () const
— const bool **getBouclier** () const
— void **setBouclier** ()
Si le ContenuTuile est une ville, permet de lui attribuer un bouclier.
— void **setType** (const TypesTuiles t)
— bool **operator==** (const ContenanceTuile &c) const
— **ContenanceTuile** (const ContenanceTuile &c)
— ContenanceTuile & **operator=** (const ContenanceTuile &c)

Amis

— class **Pioche**
— class **Tuile**

7.3.1 Documentation des constructeurs et destructeur

7.3.1.1 ContenanceTuile()

```
ContenanceTuile::ContenanceTuile (
            const TypesTuiles t,
            const int n )  [inline]
```

il prend en paramètre *nbT*.

Construteur

Paramètres

t	est de type TypesTuiles const.
n	est un entier const.

La documentation de cette classe a été générée à partir du fichier suivant :

— tuile.h

7.4 Référence de la classe Controller

Fonctions membres publiques

— Controller (int nj)

Le controller contient les principaux algorithmes permettant de gerer une partie.

— **Controller** (int nj, vector< string > listeNomJoueur, vector< int > listeNumExtensions)
— **Controller** (const Controller &)=delete
— Controller & **operator=** (const Controller &)=delete
— bool **getPaysansActive** () const
— bool **getAbbeActive** () const
— bool **getRiviereActive** () const
— bool **getAubergeActive** () const
— Espace * getEspace (const ContenanceTuile *c)

Permet de récuperer l'espace d'une contenue de tuile.

— Espace * getEspaceTuile (const ContenanceTuile &c, const vector< Espace * > e)

Permet de récuperer l'espace d'une contenue de tuile.

— void creerEspace (const Tuile *T)

Permet de regrouper les différents espaces d'une même tuile en prenant tous les cas possibles selon les règles et extensions.

— void fusionVoisin (const Tuile *tuile)

Permet de fusionner les différents espaces de deux tuiles différentes et de regrouper les mêmes espaces.

— void placementTuile (Tuile *newTuile, int x, int y)

Permet de placer la tuile sur le plateau en attribuant tous ses voisins s'il en a et ses coordonnées sur le plateau.

— void placementMeeple (Joueur *j, Meeple *m, TypesTuiles tm, int i, int x, int y)

Permet de placer un meeple sur une contenu de tuile en initialisant son type, le contenu où il est, le joueur à qui appartient le meeple, ses coordonnées, et l'ajout dans son espace, de plus on retire le meeple au joueur.

— bool placementTuileAutorise (Tuile newTuile)

Vérifie si la tuile donnée peut être autorisée n'importe où sur le plateau, à côté d'autres tuiles et selon les règles.

— bool estCompatible (Tuile newTuile, int x, int y)

Vérifie si la tuile donnée peut être placée à la position x, y sur le plateau.

— void **nextTour** ()

Permet de passer au joueur suivant, d'incrémenter le nombre de tour et de terminer la partie si elle est finie.

— void **AffichageJoueurs** ()

la méthode AffichageJoueurs permet d'afficher tous les joueurs de la partie.

— vector< Joueur * > **getJoueurs** ()
— bool **getFini** () const
— int **getNbJoueur** () const
— int **getTour** () const
— Pioche * **getPioche** () const
— int **getNumJoueurActu** () const
— Plateau * **getPlateau** () const
— vector< int > **getExtensions** () const
— ModeJeu * **getModeJeu** (int i) const
— bool **placementTuileAutorise** (Tuile newTuile, Plateau *plateau)
— bool **estCompatible** (Tuile newTuile, int x, int y, Plateau *plateau)
— bool **validationPlacementRiviere** (Tuile newTuile, int x, int y, Plateau *plateau)
— void **placementMeeple** (Joueur *j, Meeple *m, TypeMeeple tm, int i, int x, int y, Plateau *plateau)
— void **placementTuile** (Tuile *newTuile, int x, int y, Plateau *plateau)

7.4.1 Documentation des constructeurs et destructeur

7.4.1.1 Controller()

```
Controller::Controller (
            int nj )
```

Le controller contient les principaux algorithmes permettant de gerer une partie.

Constructeur

Paramètres

nj	un entier du nombre de joueur.
listeNomJoueur	un vector de string.
listeNumExtensions	un vector d'entier.

7.4.2 Documentation des fonctions membres

7.4.2.1 creerEspace()

```
void Controller::creerEspace (
            const Tuile * T )
```

Permet de regrouper les différents espaces d'une même tuile en prenant tous les cas possibles selon les règles et extensions.

Paramètres

T	

7.4.2.2 estCompatible()

```
bool Controller::estCompatible (
            Tuile newTuile,
            int x,
            int y )
```

Vérifie si la tuile donnée peut être placée à la position x, y sur le plateau.

Paramètres

newTuile	est de type Tuile.
newTuile	est de type Tuile.
x	pour la position horizontale de la tuile.
y	pour la position verticale de la tuile.

7.4.2.3 fusionVoisin()

```
void Controller::fusionVoisin (
            const Tuile * tuile )
```

Permet de fusionner les différents espaces de deux tuiles différentes et de regrouper les mêmes espaces.

Paramètres

tuile	

7.4.2.4 getEspace()

```
Espace * Controller::getEspace (
            const ContenanceTuile * c )
```

Permet de récuperer l'espace d'une contenue de tuile.

Paramètres

c	

Renvoie

7.4.2.5 getEspaceTuile()

```
Espace * Controller::getEspaceTuile (
            const ContenanceTuile & c,
            const vector< Espace * > e )
```

Permet de récuperer l'espace d'une contenue de tuile.

Paramètres

c	
e	

Renvoie

7.4.2.6 placementMeeple()

```
void Controller::placementMeeple (
            Joueur * j,
            Meeple * m,
            TypesTuiles tm,
            int i,
            int x,
            int y )
```

Permet de placer un meeple sur une contenu de tuile en initialisant son type, le contenu où il est, le joueur à qui appartient le meeple, ses coordonnées, et l'ajout dans son espace, de plus on retire le meeple au joueur.

Paramètres

j	est un pointeur de Joueur.
m	est un pointeur de Meeple.
tm	est un TypeMeeple.
i	est un entier.
x	pour la position.
y	pour la position.

7.4.2.7 placementTuile()

```
void Controller::placementTuile (
            Tuile * newTuile,
            int x,
            int y )
```

Permet de placer la tuile sur le plateau en attribuant tous ses voisins s'il en a et ses coordonnées sur le plateau.

Paramètres

newTuile	est un pointeur de Tuile.
x	est la position horizontal.
y	est la position vertical.

7.4.2.8 placementTuileAutorise()

```
bool Controller::placementTuileAutorise (
            Tuile newTuile )
```

Vérifie si la tuile donnée peut être autorisée n'importe où sur le plateau, à côté d'autres tuiles et selon les règles.

Paramètres

newTuile	est de type Tuile.

La documentation de cette classe a été générée à partir du fichier suivant :
— controller.h

7.5 Référence de la classe Espace

La classe Espace contient les différents éléments d'un Espace (ensemble de ContenuTuile de même type sur plusieurs Tuiles différentes tel qu'une ville ou champ)

```
#include <espace.h>
```

Fonctions membres publiques

— Espace (const TypesTuiles &Tt)
il prent en paramètre Tt.
— const int getNbrMeeple () const
récupérer le nombre de meeple
— const int getNbrBouclier () const
récupérer le nombre de bouclier
— const TypesTuiles & getType () const
récupérer le type
— const vector< const ContenanceTuile * > getContenus () const
récupérer l'extention
— const ContenanceTuile * **getContenus** (int i) const
— const vector< const Meeple * > getMeeples () const
getMeeples
— const Meeple * **getMeeples** (int i) const
— const size_t getNbrContenanceTuile () const
récupérer le nombre de contenance de la tuile.
— void **addBouclier** ()
ajouter un bouclier
— void addContenance (const ContenanceTuile *C)
ajouter une contenance
— void addMeeple (const Meeple *M)
ajouter un meeple.
— Espace * **fusionEspace** (Espace *e)
— bool isComplete ()
vérifie si c'est complet.
— const bool isFree () const
vérifie si la place est libre.
— void **calculScore** ()
calcul le score.

Amis

— class **Controller**

7.5.1 Description détaillée

La classe Espace contient les différents éléments d'un Espace (ensemble de ContenuTuile de même type sur plusieurs Tuiles différentes tel qu'une ville ou champ)

7.5.2 Documentation des constructeurs et destructeur

7.5.2.1 Espace()

```
Espace::Espace (
            const TypesTuiles & Tt )  [inline]
```

il prent en paramètre *Tt*.

Constructeur

Paramètres

Tt	est de type TypesTuiles.

7.5.3 Documentation des fonctions membres

7.5.3.1 addContenance()

```
void Espace::addContenance (
            const ContenanceTuile * C )
```

ajouter une contenance

Paramètres

C	est de type ContenanceTuile.

7.5.3.2 addMeeple()

```
void Espace::addMeeple (
            const Meeple * M )
```

ajouter un meeple.

Paramètres

M	est de type meeple

7.5.3.3 getContenus()

```
const vector< const ContenanceTuile * > Espace::getContenus ( ) const  [inline]
```

récupérer l'extention

Renvoie

le contenus

7.5.3.4 getMeeples()

```
const vector< const Meeple * > Espace::getMeeples ( ) const  [inline]
```

getMeeples

Renvoie

7.5.3.5 getNbrBouclier()

```
const int Espace::getNbrBouclier ( ) const  [inline]
```

récupérer le nombre de bouclier

Renvoie

nombre de bouclier

7.5.3.6 getNbrContenanceTuile()

```
const size_t Espace::getNbrContenanceTuile ( ) const  [inline]
```

récupérer le nombre de contenance de la tuile.

Renvoie

la taille de contenus

7.5.3.7 getNbrMeeple()

```
const int Espace::getNbrMeeple ( ) const  [inline]
```

récupérer le nombre de meeple

Renvoie

nombre de Meeple

7.5.3.8 getType()

```
const TypesTuiles & Espace::getType ( ) const  [inline]
```

récupérer le type

Renvoie

le type

7.5.3.9 isComplete()

```
bool Espace::isComplete ( )
```

vérifie si c'est complet.

Renvoie

vrai ou faux.

7.5.3.10 isFree()

```
const bool Espace::isFree ( ) const  [inline]
```

vérifie si la place est libre.

Renvoie

vrai ou faux

La documentation de cette classe a été générée à partir du fichier suivant :
— espace.h

7.6 Référence de la classe Joueur

Fonctions membres publiques

— Joueur (int uid, int nb, string str)
il prend en paramètre uid, nb et str
— **Joueur** (int uid)
— const int **getId** () const
— const int **getScore** () const
— const size_t **getNbrMeeplesUsed** () const
— const int **getNbrMeeples** () const
— const string **getName** () const
— const vector< const Meeple * > **getMeeples** () const
— void addScore (int pts)
ajouter des points au score.
— void addMeeple (const Meeple &m)
Attribue un meeple lorsque le joueur en question décide de placer un meeple sur une tuile, en tenant compte du nombre maximum de meeple défini par les extensions.
— void addName (const string &str)
Permet de nommer un joueur de la partie.
— void **removeMeeple** ()
déplacer un meeple
— void setNbMeeple (int i)
modifier le nombre de meeple

7.6.1 Documentation des constructeurs et destructeur

7.6.1.1 Joueur()

```
Joueur::Joueur (
            int uid,
            int nb,
            string str )  [inline]
```

il prend en paramètre *uid*, *nb* et *str*

Construteur

Paramètres

uid	est un entier
nb	est un entier
str	est de type string

7.6.2 Documentation des fonctions membres

7.6.2.1 addMeeple()

```
void Joueur::addMeeple (
            const Meeple & m )
```

Attribue un meeple lorsque le joueur en question décide de placer un meeple sur une tuile, en tenant compte du nombre maximum de meeple défini par les extensions.

Paramètres

m	

7.6.2.2 addName()

```
void Joueur::addName (
            const string & str )
```

Permet de nommer un joueur de la partie.

Paramètres

str	

7.6.2.3 addScore()

```
void Joueur::addScore (
            int pts )  [inline]
```

ajouter des points au score.

Paramètres

pts	

7.6.2.4 setNbMeeple()

```
void Joueur::setNbMeeple (
            int i )  [inline]
```

modifier le nombre de meeple

Paramètres

i	

La documentation de cette classe a été générée à partir du fichier suivant :

— joueur.h

7.7 Référence de la classe Meeple

la classe fille

```
#include <meeple.h>
```

Fonctions membres publiques

— Meeple ()
Meeple.
— **Meeple** (const Meeple &)=delete
— Meeple & **operator=** (const Meeple &)=delete
— bool **isUsed** () const
— const TypeMeeple & **getType** () const
— const int **getIdJoueur** () const
— const ContenanceTuile * **getContenanceTuile** () const
— void **setType** (const TypeMeeple &ty)
— void **setType** (const NomMeeple &name)
— void **setIdJoueur** (int i)
— void **setContenance** (ContenanceTuile &ct)

7.7.1 Description détaillée

la classe fille

7.7.2 Documentation des constructeurs et destructeur

7.7.2.1 Meeple()

```
Meeple::Meeple ( )  [inline]
```

Meeple.

Constructeur

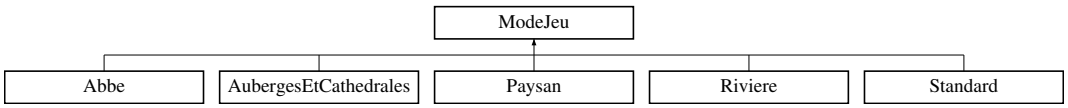
La documentation de cette classe a été générée à partir du fichier suivant :
— meeple.h

7.8 Référence de la classe ModeJeu

La classe ModeJeu implémentation du design pattern Template Method.

```
#include <modeJeu.h>
```

Graphe d'héritage de ModeJeu:



Fonctions membres publiques

— virtual bool validationPlacementT (Tuile newTuile, int x, int y, Plateau ∗plateau)=0
Vérifie le placement d'une tuile selon les règles de l'extensions.
— virtual bool validationPlacementM (const TypesTuiles &tp, Meeple ∗m, Espace ∗e)=0
Vérifie le placement d'un meeple selon les règles de l'extension.
— virtual void **affichage** ()=0

7.8.1 Description détaillée

La classe ModeJeu implémentation du design pattern Template Method.

7.8.2 Documentation des fonctions membres

7.8.2.1 validationPlacementM()

```
virtual bool ModeJeu::validationPlacementM (
            const TypesTuiles & tp,
            Meeple * m,
            Espace * e )  [pure virtual]
```

Vérifie le placement d'un meeple selon les règles de l'extension.

Paramètres

tp	
m	
e	

Renvoie

Implémenté dans Standard, Riviere, AubergesEtCathedrales, Paysan, et Abbe.

7.8.2.2 validationPlacementT()

```
virtual bool ModeJeu::validationPlacementT (
            Tuile newTuile,
            int x,
            int y,
            Plateau * plateau )  [pure virtual]
```

Vérifie le placement d'une tuile selon les règles de l'extensions.

Paramètres

newTuile	
x	
y	
plateau	

Renvoie

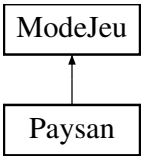
Implémenté dans Standard, Riviere, AubergesEtCathedrales, Paysan, et Abbe.

La documentation de cette classe a été générée à partir du fichier suivant :

— modeJeu.h

7.9 Référence de la classe Paysan

Graphe d'héritage de Paysan:



Fonctions membres publiques

- bool validationPlacementT (Tuile newTuile, int x, int y, Plateau ∗plateau)
  Vérifie le placement d'une tuile selon les règles de l'extensions.
- bool validationPlacementM (const TypesTuiles &tp, Meeple ∗m, Espace ∗e)
  Vérifie le placement d'un meeple selon les règles de l'extension.
- void affichage ()

7.9.1 Documentation des fonctions membres

7.9.1.1 affichage()

```
void Paysan::affichage ( )  [inline], [virtual]
```

Implémente ModeJeu.

7.9.1.2 validationPlacementM()

```
bool Paysan::validationPlacementM (
            const TypesTuiles & tp,
            Meeple * m,
            Espace * e )  [inline], [virtual]
```

Vérifie le placement d'un meeple selon les règles de l'extension.

Paramètres

tp	
m	
e	

Renvoie

Implémente ModeJeu.

7.9.1.3 validationPlacementT()

```
bool Paysan::validationPlacementT (
            Tuile newTuile,
            int x,
            int y,
            Plateau * plateau )  [inline], [virtual]
```

Vérifie le placement d'une tuile selon les règles de l'extensions.

Paramètres

newTuile	
x	
y	
plateau	

Renvoie

Implémente ModeJeu.

La documentation de cette classe a été générée à partir du fichier suivant :

— modeJeu.h

7.10 Référence de la classe Pioche

Fonctions membres publiques

— Pioche (vector< int > mode)
il prend en paramètre mode.
— const Tuile & **getTuile** (int i) const
— size_t **getNbTuiles** () const
— Tuile * piocher (int nbTour, bool riviereActive)
c'est la fonction qui relie l'action de piocher
— bool estVide () const
vérifie si l'emplacement est vide
— **Pioche** (const Pioche &)=delete
— Pioche & **operator=** (const Pioche &)=delete
— bool **getBouclier** (ContenanceTuile &c)

7.10.1 Documentation des constructeurs et destructeur

7.10.1.1 Pioche()

```
Pioche::Pioche (
            vector< int > mode )
```

il prend en paramètre *mode*.

Construteur

Paramètres

mode	est un vector d'entier.

7.10.2 Documentation des fonctions membres

7.10.2.1 estVide()

```
bool Pioche::estVide ( ) const  [inline]
```

vérifie si l'emplacement est vide

Renvoie

7.10.2.2 piocher()

```
Tuile * Pioche::piocher (
            int nbTour,
            bool riviereActive )
```

c'est la fonction qui relie l'action de piocher

Paramètres

nbTour	
riviereActive	

Renvoie

La documentation de cette classe a été générée à partir du fichier suivant :

— pioche.h

7.11 Référence de la classe Plateau

La classe Plateau permet de gérer le plateau, avec les tuiles, les espaces.

```
#include <plateau.h>
```

Fonctions membres publiques

- Plateau (int nbT)
  il prend en paramètre nbT.
- int **getNbrTuiles** () const
- int **getNbrEspaces** () const
- void **setNbrTuiles** (const int &nbrT)
- void ajouterTuiles (Tuile *tuile)
  Permet d'ajouter les tuiles dans le plateau .
- void ajouterEspace (Espace *espace)
  Permet d'ajouter un espace dans le plateau.
- void supprimerEspace (Espace *espace)
  Permet de supprimer un espace sur le plateau.
- std::vector< Tuile * > **getTuiles** () const
- std::vector< Espace * > **getEspaces** () const
- Tuile * existeTuile (int x, int y)
  Vérifie si il existe une tuile a cette emplacement.

7.11.1 Description détaillée

La classe Plateau permet de gérer le plateau, avec les tuiles, les espaces.

7.11.2 Documentation des constructeurs et destructeur

7.11.2.1 Plateau()

```
Plateau::Plateau (
            int nbT )  [inline]
```

il prend en paramètre *nbT*.

Construteur

Paramètres

nbT	est un entier.

7.11.3 Documentation des fonctions membres

7.11.3.1 ajouterEspace()

```
void Plateau::ajouterEspace (
            Espace * espace )
```

Permet d'ajouter un espace dans le plateau.

Paramètres

espace	

7.11.3.2 ajouterTuiles()

```
void Plateau::ajouterTuiles (
            Tuile * tuile )
```

Permet d'ajouter les tuiles dans le plateau .

Paramètres

tuile	

7.11.3.3 existeTuile()

```
Tuile * Plateau::existeTuile (
            int x,
            int y )  [inline]
```

Vérifie si il existe une tuile a cette emplacement.

Paramètres

x	
y	

Renvoie

7.11.3.4 supprimerEspace()

```
void Plateau::supprimerEspace (
            Espace * espace )
```

Permet de supprimer un espace sur le plateau.

Paramètres

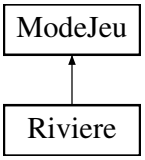
espace	

La documentation de cette classe a été générée à partir du fichier suivant :

— plateau.h

7.12 Référence de la classe Riviere

Graphe d'héritage de Riviere:



Fonctions membres publiques

— bool validationPlacementT (Tuile newTuile, int x, int y, Plateau ∗plateau)
Vérifie le placement d'une tuile selon les règles de l'extensions.
— bool validationPlacementM (const TypesTuiles &tp, Meeple ∗m, Espace ∗e)
Vérifie le placement d'un meeple selon les règles de l'extension.
— void affichage ()

7.12.1 Documentation des fonctions membres

7.12.1.1 affichage()

```
void Riviere::affichage ( )  [inline], [virtual]
```

Implémente ModeJeu.

7.12.1.2 validationPlacementM()

```
bool Riviere::validationPlacementM (
            const TypesTuiles & tp,
            Meeple * m,
            Espace * e )  [inline], [virtual]
```

Vérifie le placement d'un meeple selon les règles de l'extension.

Paramètres

tp	
m	
e	

Renvoie

Implémente ModeJeu.

7.12.1.3 validationPlacementT()

```
bool Riviere::validationPlacementT (
            Tuile newTuile,
            int x,
            int y,
            Plateau * plateau )  [inline], [virtual]
```

Vérifie le placement d'une tuile selon les règles de l'extensions.

Paramètres

newTuile	
x	
y	
plateau	

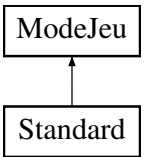
Renvoie

Implémente ModeJeu.

La documentation de cette classe a été générée à partir du fichier suivant :
— modeJeu.h

7.13 Référence de la classe Standard

Graphe d'héritage de Standard:



Fonctions membres publiques

— bool validationPlacementT (Tuile newTuile, int x, int y, Plateau ∗plateau)
Vérifie le placement d'une tuile selon les règles de l'extensions.
— bool validationPlacementM (const TypesTuiles &tp, Meeple ∗m, Espace ∗e)
Vérifie le placement d'un meeple selon les règles de l'extension.
— void affichage ()

7.13.1 Documentation des fonctions membres

7.13.1.1 affichage()

```
void Standard::affichage ( )  [inline], [virtual]
```

Implémente ModeJeu.

7.13.1.2 validationPlacementM()

```
bool Standard::validationPlacementM (
            const TypesTuiles & tp,
            Meeple * m,
            Espace * e )  [inline], [virtual]
```

Vérifie le placement d'un meeple selon les règles de l'extension.

Paramètres

tp	
m	
e	

Renvoie

Implémente ModeJeu.

7.13.1.3 validationPlacementT()

```
bool Standard::validationPlacementT (
            Tuile newTuile,
            int x,
            int y,
            Plateau * plateau )  [inline], [virtual]
```

Vérifie le placement d'une tuile selon les règles de l'extensions.

Paramètres

newTuile	
x	
y	
plateau	

Renvoie

Implémente ModeJeu.

La documentation de cette classe a été générée à partir du fichier suivant :
— modeJeu.h

7.14 Référence de la classe Tuile

Fonctions membres publiques

— Tuile (const vector< ContenanceTuile > &c)
il prend en paramètre c.
— const size_t **getSize** () const
— const int **getX** () const
— const int **getY** () const
— const Tuile * **getVoisinHaut** () const
— const Tuile * **getVoisinBas** () const
— const Tuile * **getVoisinGauche** () const
— const Tuile * **getVoisinDroite** () const
— const vector< ContenanceTuile > **getContenance** () const
— const ContenanceTuile & getContenance (int i) const
Renvoie le ContenuTuile situé au ième emplacement.
— const ContenanceTuile * **getContenancePointeur** (int i) const
— ContenanceTuile * **getContenancePointeur** (int i)
— const TypesTuiles & getContenu (int i) const
Renvoie le type du ContenuTuile situé au ième emplacement.
— void **changerOrientation** ()
— **Tuile** (const Tuile &T)
— Tuile & **operator=** (const Tuile &T)
— bool **operator==** (const Tuile &T) const
— void **setContenu** (int i, TypesTuiles c)
— void **ReplaceParChamps** ()
— void **setVoisinHaut** (Tuile *t)
— void **setVoisinBas** (Tuile *t)
— void **setVoisinGauche** (Tuile *t)
— void **setVoisinDroite** (Tuile *t)
— void **setPosX** (int x)
— void **setPosY** (int y)
— void **setX** (int x)
— void **setY** (int y)

7.14.1 Documentation des constructeurs et destructeur

7.14.1.1 Tuile()

```
Tuile::Tuile (
            const vector< ContenanceTuile > & c )
```

il prend en paramètre *c*.

Construteur

Paramètres

c	est un vector de contenanceTuile.

7.14.2 Documentation des fonctions membres

7.14.2.1 getContenance()

```
const ContenanceTuile & Tuile::getContenance (
            int i ) const  [inline]
```

Renvoie le ContenuTuile situé au ième emplacement.

Paramètres

i	

Renvoie

7.14.2.2 getContenu()

```
const TypesTuiles & Tuile::getContenu (
            int i ) const  [inline]
```

Renvoie le type du ContenuTuile situé au ième emplacement.

Paramètres

i	

Renvoie

La documentation de cette classe a été générée à partir du fichier suivant :
— tuile.h

7.15 Référence de la classe TypeMeeple

Classe mère.

```
#include <meeple.h>
```

Fonctions membres publiques

- TypeMeeple (const NomMeeple &name=rien)
  il prend en paramètre name et est initialisé à rien.
- const NomMeeple & **getNom** () const
- void **setNom** (const NomMeeple &name)

7.15.1 Description détaillée

Classe mère.

7.15.2 Documentation des constructeurs et destructeur

7.15.2.1 TypeMeeple()

```
TypeMeeple::TypeMeeple (
            const NomMeeple & name = rien )  [inline]
```

il prend en paramètre *name* et est initialisé à rien.

Construteur

Paramètres

name	est de type NomMeeple&

La documentation de cette classe a été générée à partir du fichier suivant :

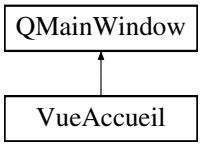
- meeple.h

7.16 Référence de la classe VueAccueil

La classe VueAccueil est de type QMainWindow.

```
#include <vueAccueil.h>
```

Graphe d'héritage de VueAccueil:



Fonctions membres publiques

- VueAccueil (QWidget *parent=nullptr)
  créer l'espace QSpinBox pour entrer le nombre de joueur, et, grâce a des checkBx, demande les extensions choisi pour le jeu.
- void **setNbrJoueur** ()
  Cette methode permet de récuperer le nombre entré dans le QSpinBox dans VueAccueil.
- int getNbrJoueur () const
  Cette fonction permet de transformer la valeur récupée par setNbJoueur et la tranformer en int()

7.16.1 Description détaillée

La classe VueAccueil est de type QMainWindow.

7.16.2 Documentation des constructeurs et destructeur

7.16.2.1 VueAccueil()

```
VueAccueil::VueAccueil (
            QWidget * parent = nullptr )  [explicit]
```

créer l'espace QSpinBox pour entrer le nombre de joueur, et, grâce a des checkBx, demande les extensions choisi pour le jeu.

Construsteur

Paramètres

parent	

7.16.3 Documentation des fonctions membres

7.16.3.1 getNbrJoueur()

```
int VueAccueil::getNbrJoueur ( ) const  [inline]
```

Cette fonction permet de transformer la valeur récupée par setNbJoueur et la tranformer en int()

Renvoie

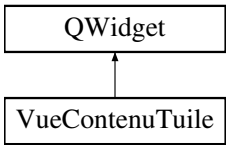
La documentation de cette classe a été générée à partir du fichier suivant :
- vueAccueil.h

7.17 Référence de la classe VueContenuTuile

La classe VueContenuTuile est un QWidget affichant un type de tuile avec couleur associe et 2 premiere lettre du type de tuile.

`#include <vueContenuTuile.h>`

Graphe d'héritage de VueContenuTuile:



Fonctions membres publiques

- VueContenuTuile (TypesTuiles type, bool bouc, QWidget ∗parent=nullptr)
  Premier constructeur sans placement de meeple. Appel de setNomCouleurSansM.
- VueContenuTuile (TypesTuiles type, bool bouc, int id, bool meeple, QWidget ∗parent=nullptr)
  Second constructeur pour placement Meeple, appel de setNomCouleurAvecM.
- void **setNomCouleurSansM** ()
  En fonction de type, défini la couleur du widget et y ajoute le bon texte.
- void setNomCouleurAvecM (const int &id)
  Meme fonctionnement que setNomCouleurSansM. rajoute le texte à l'affichage classique des 2 premières lettre pour indiquer la présence de meeple.
- TypesTuiles **getTypeTuile** () const
- bool **getBouclier** () const
- bool **getMeeple** () const
- VueContenuTuile & **operator=** (const VueContenuTuile &c)

7.17.1 Description détaillée

La classe VueContenuTuile est un QWidget affichant un type de tuile avec couleur associe et 2 premiere lettre du type de tuile.

7.17.2 Documentation des constructeurs et destructeur

7.17.2.1 VueContenuTuile() [1/2]

```
VueContenuTuile::VueContenuTuile (
            TypesTuiles type,
            bool bouc,
            QWidget * parent = nullptr )  [explicit]
```

Premier constructeur sans placement de meeple. Appel de setNomCouleurSansM.

Constructeur n°1

Paramètres

type	
bouc	
parent	

7.17.2.2 VueContenuTuile() [2/2]

```
VueContenuTuile::VueContenuTuile (
            TypesTuiles type,
            bool bouc,
            int id,
            bool meeple,
            QWidget * parent = nullptr )  [explicit]
```

Second constructeur pour placement Meeple, appel de setNomCouleurAvecM.

Constructeur n°2

Paramètres

type	
bouc	
id	
meeple	
parent	

7.17.3 Documentation des fonctions membres

7.17.3.1 setNomCouleurAvecM()

```
void VueContenuTuile::setNomCouleurAvecM (
            const int & id )
```

Meme fonctionnement que setNomCouleurSansM. rajoute le texte à l'affichage classique des 2 premières lettre pour indiquer la présence de meeple.

Paramètres

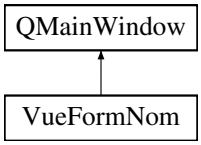
id	

La documentation de cette classe a été générée à partir du fichier suivant :

— vueContenuTuile.h

7.18 Référence de la classe VueFormNom

Graphe d'héritage de VueFormNom:



Fonctions membres publiques

- VueFormNom (int &nJoueur, vector< int > listeNumExt, QWidget *parent=nullptr)
  constructeur où sont créé les label et LineEdit, mis par la suite dans des QList pour les récupérer pour VuePartie
- int **getNbrJoueur** () const

7.18.1 Documentation des constructeurs et destructeur

7.18.1.1 VueFormNom()

```
VueFormNom::VueFormNom (
            int & nJoueur,
            vector< int > listeNumExt,
            QWidget * parent = nullptr )  [explicit]
```

constructeur où sont créé les label et LineEdit, mis par la suite dans des QList pour les récupérer pour VuePartie

Constructeur

Paramètres

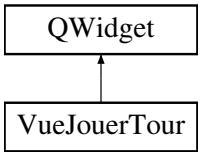
nJoueur	
listeNumExt	
parent	

La documentation de cette classe a été générée à partir du fichier suivant :

- vueFormNom.h

7.19 Référence de la classe VueJouerTour

Graphe d'héritage de VueJouerTour:



Fonctions membres publiques

— VueJouerTour (QWidget ∗parent=nullptr)
QWidget contenant un label pour designer le nom du joueur et un QLCDNumber pour designer le nombre de meeple restants.

7.19.1 Documentation des constructeurs et destructeur

7.19.1.1 VueJouerTour()

```
VueJouerTour::VueJouerTour (
            QWidget * parent = nullptr )  [explicit]
```

QWidget contenant un label pour designer le nom du joueur et un QLCDNumber pour designer le nombre de meeple restants.

Constructeur

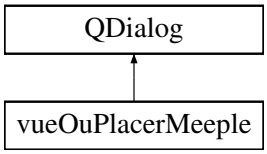
Paramètres

parent	

La documentation de cette classe a été générée à partir du fichier suivant :
— vueJouerTour.h

7.20 Référence de la classe vueOuPlacerMeeple

Graphe d'héritage de vueOuPlacerMeeple:



Fonctions membres publiques

— vueOuPlacerMeeple (int l, int c, QWidget ∗parent=nullptr, VuePartie ∗part=nullptr, Tuile ∗t=nullptr, Controller ∗con=nullptr)
vueOuPlacerMeeple
— void **setTuile** (Tuile &)

7.20.1 Documentation des constructeurs et destructeur

7.20.1.1 vueOuPlacerMeeple()

```
vueOuPlacerMeeple::vueOuPlacerMeeple (
            int l,
            int c,
            QWidget * parent = nullptr,
            VuePartie * part = nullptr,
            Tuile * t = nullptr,
            Controller * con = nullptr )  [explicit]
```

vueOuPlacerMeeple

Constructeur

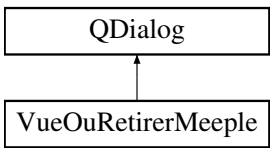
Paramètres

l	
c	
parent	
part	
t	

La documentation de cette classe a été générée à partir du fichier suivant :
— vueOuPlacerMeeple.h

7.21 Référence de la classe VueOuRetirerMeeple

Graphe d'héritage de VueOuRetirerMeeple:



Fonctions membres publiques

— VueOuRetirerMeeple (VuePartie ∗partie=nullptr, QWidget ∗parent=nullptr)
VueOuRetirerMeeple.
— int **containMeeple** (VueTuile &vt)
— void **setTuile** (Tuile &tuile, VueTuile &vt)

7.21.1 Documentation des constructeurs et destructeur

7.21.1.1 VueOuRetirerMeeple()

```
VueOuRetirerMeeple::VueOuRetirerMeeple (
            VuePartie * partie = nullptr,
            QWidget * parent = nullptr )  [explicit]
```

VueOuRetirerMeeple.

Constructeur

Paramètres

parent	
partie	

La documentation de cette classe a été générée à partir du fichier suivant :

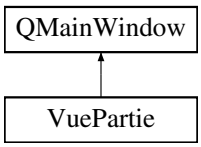
— vueOuRetirerMeeple.h

7.22 Référence de la classe VuePartie

La classe VuePartie Est decomposer en plusieurs partie : un vertical layout contenant des VueScore1Joueur. 1 layout pour y mettre la Vuetuile a placer. 1 horizontal layout contenant un label et un QLCDNumber designant numero du tour actuel. 1 horizontal layout contenant les QPushButton (valider, fin Partie etc). 1 QWidget contenant un QLabel poru designer le nom du joueur devant jouer ainsi qu'un QLCDNumber designant le nombre de meeple lui restant.

`#include <vuePartie.h>`

Graphe d'héritage de VuePartie:



Fonctions membres publiques

— VuePartie (Controller ∗c, QWidget ∗parent=nullptr)
constructeur appellant setAffichageScore, setAffichageTuile, setJoueurActu, setPlateau
— void **setAffichageScore** ()
Initialise le vertical layout dedie au informations des joueurs : converti le vecteur de Joueur fourni par le controlleur en vecteur de VueScore1Joueur. Affiche le numero du tour actuel.
— void **setAffichageTuile** ()
affiche la tuile a placer en bas a droite de la fenetre.
— void **setJoueurActu** ()
Récupère la string du joueur devant jouer via le controller, la transforme en QString via fromStdString et l'ajoute au label. recupere egalement le nombre de meeple restant au joueur et l'attribut au QLCDNumber.
— void **setPlateau** ()
initialise le plateau : centrage, affichage de la premiere tuile piochée
— void **piocherCarte** ()
— void placerMeeple (const int Nligne, const int NCol, Tuile ∗tuile)
placerMeeple
— void **retirerMeeple** ()
— void placerTuile (const int Nligne, const int NCol, Tuile ∗tuile)
Contruit et ajoute la VueTuile à partir d'une tuile du modele, a la position donnee en parametre de la fonction, sur le plateau.
— void **updateVueTuileAddM** (int l, int c, int p, Tuile ∗T)
— void **updateVueTuileRemoveM** (int l, int c)
— void **partieFinie** ()
supprime l'ensemble des widget non nécessaire pour le décompte des points

Attributs publics

— Tuile * **tuilePlace**
— VueTuile * **vueTuilePlace**

7.22.1 Description détaillée

La classe VuePartie Est decomposer en plusieurs partie : un vertical layout contenant des VueScore1Joueur. 1 layout pour y mettre la Vuetuile a placer. 1 horizontal layout contenant un label et un QLCDNumber designant numero du tour actuel. 1 horizontal layout contenant les QPushButton (valider, fin Partie etc). 1 QWidget contenant un QLabel poru designer le nom du joueur devant jouer ainsi qu'un QLCDNumber designant le nombre de meeple lui restant.

7.22.2 Documentation des constructeurs et destructeur

7.22.2.1 VuePartie()

```
VuePartie::VuePartie (
            Controller * c,
            QWidget * parent = nullptr )  [explicit]
```

constructeur appellant setAffichageScore, setAffichageTuile, setJoueurActu, setPlateau

Constructeur

Paramètres

c	
parent	

7.22.3 Documentation des fonctions membres

7.22.3.1 placerMeeple()

```
void VuePartie::placerMeeple (
            const int Nligne,
            const int NCol,
            Tuile * tuile )
```

placerMeeple

Paramètres

Nligne	
NCol	
tuile	

7.22.3.2 placerTuile()

```
void VuePartie::placerTuile (
            const int Nligne,
            const int NCol,
            Tuile * tuile )
```

Contruit et ajoute la VueTuile à partir d'une tuile du modele, a la position donnee en parametre de la fonction, sur le plateau.

Paramètres

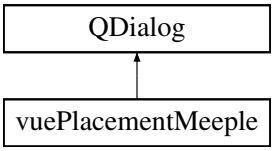
Nligne	
NCol	
tuile	

La documentation de cette classe a été générée à partir du fichier suivant :

— vuePartie.h

7.23 Référence de la classe vuePlacementMeeple

Graphe d'héritage de vuePlacementMeeple:



Fonctions membres publiques

— vuePlacementMeeple (QWidget ∗parent=nullptr, VuePartie ∗partie=nullptr, const int l=0, const int c=0, Tuile ∗tuile=nullptr, Controller ∗con=nullptr)
vuePlacementMeeple
— int getNligne () const
getNligne
— int getNCol () const
getNCol

7.23.1 Documentation des constructeurs et destructeur

7.23.1.1 vuePlacementMeeple()

```
vuePlacementMeeple::vuePlacementMeeple (
            QWidget * parent = nullptr,
            VuePartie * partie = nullptr,
            const int l = 0,
            const int c = 0,
            Tuile * tuile = nullptr,
            Controller * con = nullptr )  [explicit]
```

vuePlacementMeeple

Constructeur

Paramètres

parent	
partie	
l	
c	
tuile	

7.23.2 Documentation des fonctions membres

7.23.2.1 getNCol()

```
int vuePlacementMeeple::getNCol ( ) const  [inline]
```

getNCol

Renvoie

Ncol

7.23.2.2 getNligne()

```
int vuePlacementMeeple::getNligne ( ) const  [inline]
```

getNligne

Renvoie

Nligne

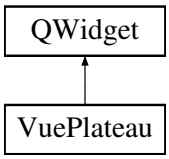
La documentation de cette classe a été générée à partir du fichier suivant :
— vuePlacementMeeple.h

7.24 Référence de la classe VuePlateau

La classe VuePlateau est un Qwidget contenant un QWidgetTable.

`#include <vuePlateau.h>`

Graphe d'héritage de VuePlateau:



Fonctions membres publiques

- VuePlateau (QWidget ∗parent=nullptr)
  Appel de setTable.
- void **setTable** ()
  Permet de centrer la plateau sur la case 72, 72.

7.24.1 Description détaillée

La classe VuePlateau est un Qwidget contenant un QWidgetTable.

7.24.2 Documentation des constructeurs et destructeur

7.24.2.1 VuePlateau()

```
VuePlateau::VuePlateau (
            QWidget * parent = nullptr )  [explicit]
```

Appel de setTable.

Constructeur

Paramètres

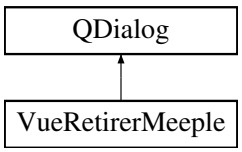
parent	

La documentation de cette classe a été générée à partir du fichier suivant :

- vuePlateau.h

7.25 Référence de la classe VueRetirerMeeple

Graphe d'héritage de VueRetirerMeeple:



Fonctions membres publiques

— VueRetirerMeeple (VuePartie *partie=nullptr, QWidget *parent=nullptr)
VueRetirerMeeple.

7.25.1 Documentation des constructeurs et destructeur

7.25.1.1 VueRetirerMeeple()

```
VueRetirerMeeple::VueRetirerMeeple (
            VuePartie * partie = nullptr,
            QWidget * parent = nullptr )  [explicit]
```

VueRetirerMeeple.

Constructeur

Paramètres

parent	
partie	

La documentation de cette classe a été générée à partir du fichier suivant :

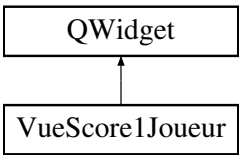
— vueRetirerMeeple.h

7.26 Référence de la classe VueScore1Joueur

La classe VueScore1Joueur est un QWidget constitué d'un vertical layout contenant 2 QLabel pour désigner le nom du joueur et le score, 1 QLCDNumber pour afficher le score et 2 boutton dans un vertical layout pour changer le score.

```
#include <vueScore1Joueur.h>
```

Graphe d'héritage de VueScore1Joueur:



Fonctions membres publiques

— VueScore1Joueur (std::string nomJ, int id, QWidget ∗parent=nullptr)
Transforme l'id du joueur en std::string et transforme cette string en QString afin de l'ajouter au label.

7.26.1 Description détaillée

La classe VueScore1Joueur est un QWidget constitué d'un vertical layout contenant 2 QLabel pour désigner le nom du joueur et le score, 1 QLCDNumber pour afficher le score et 2 boutton dans un vertical layout pour changer le score.

7.26.2 Documentation des constructeurs et destructeur

7.26.2.1 VueScore1Joueur()

```
VueScore1Joueur::VueScore1Joueur (
            std::string nomJ,
            int id,
            QWidget * parent = nullptr )  [explicit]
```

Transforme l'id du joueur en std::string et transforme cette string en QString afin de l'ajouter au label.

Constructeur

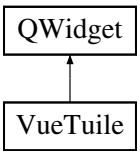
Paramètres

nomJ	
id	
parent	

La documentation de cette classe a été générée à partir du fichier suivant :
— vueScore1Joueur.h

7.27 Référence de la classe VueTuile

Graphe d'héritage de VueTuile:



Fonctions membres publiques

- VueTuile (Tuile *tuile, QWidget *parent=nullptr)
  Construit une Vuetuile à partir d'une tuile du modele. Utilise setContenuTuile.
- VueTuile (Tuile *tuile, int p, QWidget *parent=nullptr)
  Meme fonction que le constructeur 1 mais on ajoute le placement d'un meeple. Utilise ajoutMeeple.
- void setContenuTuile (Tuile &tuile)
  Pour chaque case de la tuile, creer la vueContenuTuile associé et l'ajoute au bonne emplacement sur la QGrid.
- void addMeeple (Tuile *tuile, int p, int id)
  ajoute un meeple
- vector< VueContenuTuile * > & getVueContenuT ()
  getVueContenuT

7.27.1 Documentation des constructeurs et destructeur

7.27.1.1 VueTuile() [1/2]

```
VueTuile::VueTuile (
            Tuile * tuile,
            QWidget * parent = nullptr )  [explicit]
```

Construit une Vuetuile à partir d'une tuile du modele. Utilise setContenuTuile.

Constructeur n°1

Paramètres

tuile	
parent	

7.27.1.2 VueTuile() [2/2]

```
VueTuile::VueTuile (
            Tuile * tuile,
            int p,
            QWidget * parent = nullptr )  [explicit]
```

Meme fonction que le constructeur 1 mais on ajoute le placement d'un meeple. Utilise ajoutMeeple.

Paramètres

tuile	
p	
parent	

7.27.2 Documentation des fonctions membres

7.27.2.1 addMeeple()

```
void VueTuile::addMeeple (
            Tuile * tuile,
            int p,
            int id )
```

ajoute un meeple

Paramètres

tuile	
p	

7.27.2.2 getVueContenuT()

```
vector< VueContenuTuile * > & VueTuile::getVueContenuT ( )  [inline]
```

getVueContenuT

Renvoie

7.27.2.3 setContenuTuile()

```
void VueTuile::setContenuTuile (
            Tuile & tuile )
```

Pour chaque case de la tuile, creer la vueContenuTuile associé et l'ajoute au bonne emplacement sur la QGrid.

Paramètres

tuile	

La documentation de cette classe a été générée à partir du fichier suivant :

— vueTuile.h

Chapitre 8

Documentation des fichiers

8.1 Référence du fichier controller.h

```
#include <iostream>
#include <string>
#include <vector>
#include "pioche.h"
#include "joueur.h"
#include "plateau.h"
#include "tuile.h"
#include "meeple.h"
#include "modeJeu.h"
```

Classes

— class Controller

Espaces de nommage

— namespace std

8.2 controller.h

Aller à la documentation de ce fichier.

```
1 #ifndef _CONTROLLER_H
2 #define _CONTROLLER_H
3 #include <iostream>
4 #include <string>
5 #include <vector>
6 #include "pioche.h"
7 #include "joueur.h"
8 #include "plateau.h"
9 #include "tuile.h"
10 #include "meeple.h"
11 #include "modeJeu.h"
12
28 using namespace std;
29
30 class Controller{
```

```
31
32
33      private:
34
35          bool fini;
36          vector<Joueur*> listeJoueurs;
37          vector<int> extensions;
38          ModeJeu* modeJeu[5];
39          Plateau *plateau;
40          Pioche *pioche;
41          bool riviereActive =false;
42          bool paysansActive = false;
43          bool aubergeActive = false;
44          bool abbeActive = false;
45          int tour;
46          int nbJoueurs;
47          int numJoueurActu; // Va de 0 à nbrJoueur - 1
48
49      public:
50
51      Controller() = default;
52
60          Controller(int nj);
61          Controller(int nj,vector<string> listeNomJoueur,vector<int> listeNumExtensions);
62          //Destructeur
63          ~Controller() = default;
64          Controller(const Controller&) = delete;
65          Controller& operator=(const Controller&) = delete ;
66
67          inline bool getPaysansActive() const {return paysansActive;}
68          inline bool getAbbeActive() const {return abbeActive;}
69          inline bool getRiviereActive() const {return riviereActive;}
70          inline bool getAubergeActive() const {return aubergeActive;}
71
77       Espace* getEspace(const ContenanceTuile* c);
78
85      Espace* getEspaceTuile(const ContenanceTuile& c,const vector<Espace*> e);
86
91      void creerEspace(const Tuile* T);
92
97      void fusionVoisin(const Tuile* tuile);
98
99
106           void placementTuile(Tuile *newTuile,int x,int y);
107
117           void placementMeeple(Joueur* j,Meeple* m,TypesTuiles tm,int i,int x,int y);
118
123           bool placementTuileAutorise(Tuile newTuile);
124
125
126           //Vérifie si la tuile donnée peut être placée à la position x, y sur le plateau
134           bool estCompatible(Tuile newTuile,int x,int y);
135          // void compteScore(State s);
136          // void compteScore(TypesTuiles t, State s);
137
141           void nextTour();
142           //void compteScoreAbbaye(State s);
143
147           void AffichageJoueurs(){
148               for(int i=0;i<listeJoueurs.size();i++){
149                   cout « "Joueur " « i « " : " « listeJoueurs.at(i)->getNbrMeeples() « " meeples" « endl;
150               }
151           }
152
153           vector<Joueur*> getJoueurs(){
154               return listeJoueurs;
155           }
156
157           //Getters
158           inline bool getFini() const{
159               return this->fini;
160           }
161
162           inline int getNbJoueur() const{
163               return this->nbJoueurs;
164           }
165
166
167           inline int getTour() const{
168               return this->tour;
169           }
170
171            inline Pioche* getPioche() const{return this->pioche;}
172
173
174            inline int getNumJoueurActu() const{return this->numJoueurActu;}
175            inline Plateau* getPlateau() const{
```

```
176             return this->plateau;
177         }
178
179             inline vector<int> getExtensions() const{
180             return this->extensions;
181         }
182
183         //retourner le modedejeu
184         inline ModeJeu* getModeJeu(int i) const{
185             return this->modeJeu[i];
186         }
187
188     //Fonction test
189     bool placementTuileAutorise(Tuile newTuile,Plateau* plateau);
190     bool estCompatible(Tuile newTuile,int x,int y,Plateau *plateau);
191     bool validationPlacementRiviere(Tuile newTuile,int x,int y,Plateau *plateau);
192     void placementMeeple(Joueur* j,Meeple* m,TypeMeeple tm,int i,int x,int y,Plateau *plateau);
193     void placementTuile(Tuile *newTuile,int x,int y,Plateau *plateau);
194 };
195
196 #endif
```

8.3 Référence du fichier espace.h

```
#include <stdio.h>
#include "meeple.h"
#include <vector>
```

Classes

— class Espace

La classe Espace contient les différents éléments d'un Espace (ensemble de ContenuTuile de même type sur plusieurs Tuiles différentes tel qu'une ville ou champ)

Fonctions

— ostream & **operator**<< (ostream &f, const Espace &E)

8.4 espace.h

Aller à la documentation de ce fichier.

```
1
2
3 #ifndef espace_h
4 #define espace_h
5
6 #include <stdio.h>
7 #include "meeple.h"
8 #include <vector>
9
18 class Espace {
19
20     friend class Controller;
21
22 private :
23
24     TypesTuiles type;
25     int nbrBouclier;
26     int nbrMeeple;
27     vector<const ContenanceTuile*> contenus;
28     vector<const Meeple*> meeples;
29
30 public :
31
```

```
37     Espace(const TypesTuiles& Tt): nbrBouclier(0), nbrMeeple(0), type(Tt){}
38
39
44     inline const int getNbrMeeple() const { return nbrMeeple; }
45
50     inline const int getNbrBouclier() const { return nbrBouclier; }
51
56     inline const TypesTuiles& getType() const { return type; }
57
62     inline const vector<const ContenanceTuile*> getContenus() const { return contenus; }
63
64     inline const ContenanceTuile* getContenus(int i) const { return contenus[i]; }
65
66
71     inline const vector<const Meeple*> getMeeples() const { return meeples; }
72
73     inline const Meeple* getMeeples(int i) const { return meeples[i]; }
74
79     inline const size_t getNbrContenanceTuile() const { return contenus.size(); }
80
84     inline void addBouclier() { nbrBouclier += 1; }
85
90     void addContenance(const ContenanceTuile* C);
91
92
97     void addMeeple(const Meeple* M);
98
99     Espace* fusionEspace(Espace* e);
100
105     bool isComplete();
106
111     inline const bool isFree() const {return meeples.size() == 0;}
112
116     void calculScore();
117 };
118
119
120 ostream& operator<<(ostream& f, const Espace& E);
121
122
123 #endif
```

8.5 Référence du fichier joueur.h

```
#include <stdio.h>
#include <iostream>
#include <vector>
#include <string>
#include "meeple.h"
```

Classes

— class Joueur

Fonctions

— ostream & **operator**<< (ostream &f, const Joueur &J)

8.6 joueur.h

Aller à la documentation de ce fichier.

```
1
2 #ifndef joueur_h
3 #define joueur_h
4
5
6 #include <stdio.h>
7 #include <iostream>
8 #include <vector>
9 #include <string>
10 #include "meeple.h"
11
12
18 class Joueur {
19
20 private:
21
22     int id;
23     int score;
24     int nbrMeeples;
25     string name;
26     vector<const Meeple*> meeples;
27
28 public:
29
37     Joueur(int uid, int nb, string str) : id(uid), nbrMeeples(nb), score(0), name(str), meeples(NULL) {}
38     Joueur(int uid) : id(uid){}
39
40     inline const int getId() const { return id; }
41     inline const int getScore() const { return score; }
42     inline const size_t getNbrMeeplesUsed() const { return meeples.size(); }
43     inline const int getNbrMeeples() const { return nbrMeeples; }
44     inline const string getName() const { return name; }
45     inline const vector<const Meeple*> getMeeples() const { return meeples; }
46
51     inline void addScore(int pts) { score += pts; }
52
57     void addMeeple(const Meeple& m);
58
63     void addName(const string& str);
64
68     void removeMeeple(){nbrMeeples--;}
69
74     void setNbMeeple(int i){
75         nbrMeeples = i;
76     }
77 };
78
79
80 ostream& operator<<(ostream& f, const Joueur& J);
81
82
83 #endif
```

8.7 meeple.h

```
1
2 #ifndef meeple_h
3 #define meeple_h
4
5
6 #include <stdio.h>
7 #include <iostream>
8 #include "tuile.h"
9
10
11
12 using namespace std;
13
14
15
20 enum NomMeeple {
21     chevalier,
22     paysan,
23     abbe,
24     voleur,
25     rien
26 };
27
```

```
28
29
34 static const char* NomMeeple_str[] = { "chevalier", "paysan", "abbe", "voleur" };
35
36
37
38
42 class TypeMeeple {
43
44     private:
45
46         NomMeeple nom;
47
48     public:
49
55         TypeMeeple(const NomMeeple& name=rien): nom(name){}  // constructeur
56
57         inline const NomMeeple& getNom() const { return nom; }
58         inline void setNom(const NomMeeple& name) { nom = name; }
59 };
60
61
62
63 ostream& operator<<(ostream& f, const TypeMeeple& Tm);
64
65
66
67
71 class Meeple {
72
73     private:
74
75         ContenanceTuile* contenanceTuile;
76         TypeMeeple type;
77         int id_joueur;
78
79     public:
80
85         Meeple(): contenanceTuile(nullptr) {}
86
87         Meeple(const Meeple&) = delete;
88         Meeple& operator=(const Meeple&) = delete;
89
90         inline bool isUsed() const { return contenanceTuile != nullptr; } // 1 si nullptr 0 sinon
91         inline const TypeMeeple& getType() const { return type; }
92         inline const int getIdJoueur() const { return id_joueur; }
93         inline const ContenanceTuile* getContenanceTuile() const { return contenanceTuile; }
94
95         inline void setType(const TypeMeeple& ty) { type = ty; };
96         inline void setType(const NomMeeple& name) { type.setNom(name); };
97         inline void setIdJoueur(int i) { id_joueur = i; }
98         inline void setContenance(ContenanceTuile& ct) { contenanceTuile = &ct; }
99
100 };
101
102
103 ostream& operator<<(ostream& f, const Meeple& M);
104
105
106 #endif
107
```

8.8 modeJeu.h

```
1 #ifndef _MODEJEU_H
2 #define _MODEJEU_H
3
4 #include <iostream>
5 #include <string>
6 #include <vector>
7 #include "tuile.h"
8 #include "meeple.h"
9 #include "espace.h"
10 #include  "plateau.h"
11
12
30 using namespace std;
31
32
36 class ModeJeu{
37
38     public:
39
```

```
48          virtual bool validationPlacementT(Tuile newTuile,int x,int y,Plateau *plateau) = 0;
49
57          virtual bool validationPlacementM(const TypesTuiles& tp,Meeple *m,Espace *e) = 0;
58          virtual void affichage() = 0;
59
60  };
61
62  // ------------------------ EXTENSION STANDARD -------------------------------------
63
64  class Standard : public ModeJeu{
65      public:
66
67          bool validationPlacementT(Tuile newTuile,int x,int y,Plateau *plateau){
68
69           if(!plateau->existeTuile(x,y-1) && !plateau->existeTuile(x,y+1) && !plateau->existeTuile(x-1,y)
        && !plateau->existeTuile(x+1,y)){
70               return false;
71           }
72
73          if(plateau->existeTuile(x,y-1)){
74              if(newTuile.getContenu(1)!=plateau->existeTuile(x,y-1)->getContenu(5)){
75                  return false;
76              }
77          }
78
79
80          //Voisin du bas
81
82          if(plateau->existeTuile(x,y+1)){
83              if(newTuile.getContenu(5)!=plateau->existeTuile(x,y+1)->getContenu(1)){
84                  return false;
85              }
86          }
87
88          //Voisin de gauche
89          if(plateau->existeTuile(x-1,y)){
90              if(newTuile.getContenu(7)!=plateau->existeTuile(x-1,y)->getContenu(3)){
91                  return false;
92              }
93          }
94
95          //Voisin de droite
96          if(plateau->existeTuile(x+1,y)){
97              if(newTuile.getContenu(3)!=plateau->existeTuile(x+1,y)->getContenu(7)){
98                  return false;
99              }
100          }
101
102          return true;
103
104      }
105          bool validationPlacementM(const TypesTuiles& tp,Meeple *m ,Espace *e){
106
107              if(e==NULL){
108                  return false;
109              }
110
111              if(!e->isFree() || tp==TypesTuiles::champs){
112                  return false;
113              }
114              return true;
115          }
116
117          void affichage(){
118              cout<<"*****************************Standard*****************************************"<<endl;
119          }
120  };
121
122
123
124  // ------------------------ EXTENSION RIVIERE -------------------------------------
125
126  class Riviere : public ModeJeu{
127
128      public:
129
130  bool validationPlacementT(Tuile newTuile,int x,int y,Plateau *plateau){
131
132                  if((newTuile.getContenu(0)!=TypesTuiles::riviere) &&
133                   (newTuile.getContenu(1)!=TypesTuiles::riviere) &&
134                   (newTuile.getContenu(2)!=TypesTuiles::riviere) &&
135                   (newTuile.getContenu(3)!=TypesTuiles::riviere) &&
136                   (newTuile.getContenu(4)!=TypesTuiles::riviere) &&
137                   (newTuile.getContenu(5)!=TypesTuiles::riviere) &&
138                   (newTuile.getContenu(6)!=TypesTuiles::riviere) &&
139                   (newTuile.getContenu(7)!=TypesTuiles::riviere) &&
140                   (newTuile.getContenu(8)!=TypesTuiles::riviere)){ return true; }
```

```
141
142
143         if(!plateau->existeTuile(x,y-1) && !plateau->existeTuile(x,y+1) && !plateau->existeTuile(x-1,y)
    && !plateau->existeTuile(x+1,y)){
144             return false;
145         }
146
147         //Voisin du haut
148         if(plateau->existeTuile(x,y-1)){
149                 if(newTuile.getContenu(1)!=plateau->existeTuile(x,y-1)->getContenu(5)
150                 || (newTuile.getContenu(1)!=TypesTuiles::riviere)
151                 || (plateau->existeTuile(x,y-1)->getContenu(5)!=TypesTuiles::riviere)
152                 || (newTuile.getContenu(1)==plateau->existeTuile(x,y-1)->getContenu(5)
153                     && newTuile.getContenu(3)==plateau->existeTuile(x,y-1)->getContenu(3)
154                     && newTuile.getContenu(3)==TypesTuiles::riviere
155                     && newTuile.getContenu(1)==TypesTuiles::riviere)
156                 || (newTuile.getContenu(1)==plateau->existeTuile(x,y-1)->getContenu(5)
157                     && newTuile.getContenu(7)==plateau->existeTuile(x,y-1)->getContenu(7)
158                     && newTuile.getContenu(7)==TypesTuiles::riviere
159                     && newTuile.getContenu(1)==TypesTuiles::riviere)){
160                     return false;
161                 }
162
163         }
164
165         //Voisin du bas
166
167         if(plateau->existeTuile(x,y+1)){
168                 if(newTuile.getContenu(5)!=plateau->existeTuile(x,y+1)->getContenu(1)
169                 || (newTuile.getContenu(5)!=TypesTuiles::riviere)
170                 || (plateau->existeTuile(x,y+1)->getContenu(1)!=TypesTuiles::riviere)
171                 || (newTuile.getContenu(5)==plateau->existeTuile(x,y+1)->getContenu(1)
172                     && newTuile.getContenu(3)==plateau->existeTuile(x,y+1)->getContenu(3)
173                     && newTuile.getContenu(5)==TypesTuiles::riviere
174                     && newTuile.getContenu(7)==TypesTuiles::riviere)
175                 || (newTuile.getContenu(5)==plateau->existeTuile(x,y+1)->getContenu(1)
176                     && newTuile.getContenu(7)==plateau->existeTuile(x,y+1)->getContenu(7)
177                     && newTuile.getContenu(5)==TypesTuiles::riviere
178                     && newTuile.getContenu(3)==TypesTuiles::riviere)){
179                     return false;
180
181             }
182         }
183
184         //Voisin de gauche
185         if(plateau->existeTuile(x-1,y)){
186                 if(newTuile.getContenu(7)!=plateau->existeTuile(x-1,y)->getContenu(3)
187                 || (newTuile.getContenu(7)!=TypesTuiles::riviere)
188                 || (plateau->existeTuile(x-1,y)->getContenu(3)!=TypesTuiles::riviere)
189                 || (newTuile.getContenu(7)==plateau->existeTuile(x-1,y)->getContenu(3)
190                     && newTuile.getContenu(1)==plateau->existeTuile(x-1,y)->getContenu(1)
191                     && newTuile.getContenu(7)==TypesTuiles::riviere
192                     && newTuile.getContenu(1)==TypesTuiles::riviere)
193                 || (newTuile.getContenu(7)==plateau->existeTuile(x-1,y)->getContenu(3)
194                     && newTuile.getContenu(5)==plateau->existeTuile(x-1,y)->getContenu(5)
195                     && newTuile.getContenu(5)==TypesTuiles::riviere
196                     && newTuile.getContenu(7)==TypesTuiles::riviere)){
197                     return false;
198
199             }
200         }
201
202         //Voisin de droite
203         if(plateau->existeTuile(x+1,y)){
204                 if(newTuile.getContenu(3)!=plateau->existeTuile(x+1,y)->getContenu(7)
205                 || (newTuile.getContenu(3)!=TypesTuiles::riviere)
206                 || (plateau->existeTuile(x+1,y)->getContenu(7)!=TypesTuiles::riviere)
207                 || (newTuile.getContenu(3)==plateau->existeTuile(x+1,y)->getContenu(7)
208                     && newTuile.getContenu(1)==plateau->existeTuile(x+1,y)->getContenu(1)
209                     && newTuile.getContenu(3)==TypesTuiles::riviere
210                     && newTuile.getContenu(1)==TypesTuiles::riviere)
211                 || (newTuile.getContenu(3)==plateau->existeTuile(x+1,y)->getContenu(7)
212                     && newTuile.getContenu(5)==plateau->existeTuile(x+1,y)->getContenu(5)
213                     && newTuile.getContenu(3)==TypesTuiles::riviere
214                     && newTuile.getContenu(5)==TypesTuiles::riviere)){
215                     return false;
216                 }
217
218         }
219
220         return true;
221
222     }
223
224     bool validationPlacementM(const TypesTuiles& tp,Meeple *m ,Espace *e){
225         if(e==NULL){
226             return false;
```

```
227         }
228         if(!e->isFree() || tp==TypesTuiles::riviere){
229 
230             return false;
231         }
232         return true;
233     }
234 
235     void affichage(){
236         cout « " Riviere " « endl;
237     }
238 };
239 
240 //EXTENSION AUBERGES ET CATHEDRALES
241 
242 class AubergesEtCathedrales:public ModeJeu{
243     public:
244         bool validationPlacementT(Tuile newTuile,int x,int y,Plateau *plateau){
245 
246 
247 
248          if(!plateau->existeTuile(x,y-1) && !plateau->existeTuile(x,y+1) && !plateau->existeTuile(x-1,y)
     && !plateau->existeTuile(x+1,y)){
249              return false;
250          }
251 
252         if(plateau->existeTuile(x,y-1)){
253             if(newTuile.getContenu(1)!=plateau->existeTuile(x,y-1)->getContenu(5)){
254                 return false;
255             }
256         }
257 
258         //Voisin du bas
259 
260         if(plateau->existeTuile(x,y+1)){
261             if(newTuile.getContenu(5)!=plateau->existeTuile(x,y+1)->getContenu(1)){
262                 return false;
263             }
264         }
265 
266         //Voisin de gauche
267         if(plateau->existeTuile(x-1,y)){
268             if(newTuile.getContenu(7)!=plateau->existeTuile(x-1,y)->getContenu(3)){
269                 return false;
270             }
271         }
272 
273         //Voisin de droite
274         if(plateau->existeTuile(x+1,y)){
275             if(newTuile.getContenu(3)!=plateau->existeTuile(x+1,y)->getContenu(7)){
276                 return false;
277             }
278         }
279 
280         return true;
281 
282     }
283         // un meeple ne peut pas occuper une auberge ou une cathédrale
284 
285         bool validationPlacementM(const TypesTuiles& tp,Meeple *m ,Espace *e){
286             if(e==NULL){
287                 return false;
288             }
289 
290             if(!e->isFree() || tp==TypesTuiles::auberge || tp==TypesTuiles::cathedrale){
291                 return false;
292             }
293             return true;
294         }
295 
296         void affichage(){
297             cout«"AubergesEtCathedrales"«endl;
298         }
299 };
300 
301 
302 // ------------------------ EXTENSION PAYSAN --------------------------------------
303 
304 class Paysan: public ModeJeu{
305     public:
306 
307         bool validationPlacementT(Tuile newTuile,int x,int y,Plateau *plateau){
308 
309 
310          if(!plateau->existeTuile(x,y-1) && !plateau->existeTuile(x,y+1) && !plateau->existeTuile(x-1,y)
     && !plateau->existeTuile(x+1,y)){
311              return false;
```

```
312          }
313
314
315         if(plateau->existeTuile(x,y-1)){
316             if(newTuile.getContenu(1)!=plateau->existeTuile(x,y-1)->getContenu(5)){
317                 return false;
318             }
319         }
320
321
322         //Voisin du bas
323
324         if(plateau->existeTuile(x,y+1)){
325             if(newTuile.getContenu(5)!=plateau->existeTuile(x,y+1)->getContenu(1)){
326                 return false;
327             }
328         }
329
330         //Voisin de gauche
331         if(plateau->existeTuile(x-1,y)){
332             if(newTuile.getContenu(7)!=plateau->existeTuile(x-1,y)->getContenu(3)){
333                 return false;
334             }
335         }
336
337         //Voisin de droite
338         if(plateau->existeTuile(x+1,y)){
339             if(newTuile.getContenu(3)!=plateau->existeTuile(x+1,y)->getContenu(7)){
340                 return false;
341             }
342         }
343
344         return true;
345
346     }
347
348         bool validationPlacementM(const TypesTuiles& tp,Meeple *m ,Espace *e){
349             if(e==NULL){
350                 return false;
351             }
352             if(!e->isFree()) {
353                 return false;
354             }
355             return true;
356         }
357
358         void affichage(){
359             cout<<"Paysan"<<endl;
360         }
361 };
362
363
364
365 // ------------------------ EXTENSION ABBE -------------------------------------
366
367
368 class Abbe : public ModeJeu{
369     public:
370
371         bool validationPlacementT(Tuile newTuile,int x,int y,Plateau *plateau){
372
373          if(!plateau->existeTuile(x,y-1) && !plateau->existeTuile(x,y+1) && !plateau->existeTuile(x-1,y)
    && !plateau->existeTuile(x+1,y)){
374              return false;
375          }
376
377
378         if(plateau->existeTuile(x,y-1)){
379             if(newTuile.getContenu(1)!=plateau->existeTuile(x,y-1)->getContenu(5)){
380                 return false;
381             }
382         }
383
384
385         //Voisin du bas
386
387         if(plateau->existeTuile(x,y+1)){
388             if(newTuile.getContenu(5)!=plateau->existeTuile(x,y+1)->getContenu(1)){
389                 return false;
390             }
391         }
392
393         //Voisin de gauche
394         if(plateau->existeTuile(x-1,y)){
395             if(newTuile.getContenu(7)!=plateau->existeTuile(x-1,y)->getContenu(3)){
396                 return false;
397             }
```

```
398         }
399
400         //Voisin de droite
401         if(plateau->existeTuile(x+1,y)){
402             if(newTuile.getContenu(3)!=plateau->existeTuile(x+1,y)->getContenu(7)){
403                 return false;
404             }
405         }
406
407         return true;
408
409     }
410
411        bool validationPlacementM(const TypesTuiles& tp,Meeple *m ,Espace *e){
412            if(e==NULL){
413                return false;
414            }
415
416             //Si un meeple est déjà présent sur l'espace
417             if(!e->isFree()){
418                 return false;
419             }
420
421             //L'abbé ne peut être posé que sur une abbaye ou un jardin
422             if(m->getType().getNom()==NomMeeple::abbe){
423                 if(tp==TypesTuiles::abbaye || tp==TypesTuiles::jardin){
424                     return true;
425                 }
426                 else{
427                     return false;
428                 }
429             }
430             //le meeple peut être posé sur n'importe ou sauf sur le jardin
431             else{
432                 if(tp==TypesTuiles::jardin){
433                     return false;
434                 }
435                 else{
436                     return true;
437                 }
438             }
439             return true;
440         }
441         void affichage(){
442             cout«"Abbe"«endl;
443         }
444 };
445
446 #endif
```

8.9 Référence du fichier pioche.h

```
#include <stdio.h>
#include "tuile.h"
```

Classes

— class Pioche

Fonctions

— ostream & **operator**<< (ostream &f, const Pioche &T)

8.10 pioche.h

Aller à la documentation de ce fichier.

```
1
2 #ifndef pioche_h
3 #define pioche_h
4
5
6 #include <stdio.h>
7 #include "tuile.h"
8
9
15 class Pioche {
16
17 private :
18
19     vector<Tuile> tuiles;
20
21 public :
22
28     Pioche(vector<int> mode);
29
30     inline const Tuile& getTuile(int i) const {return tuiles[i];}
31     inline size_t getNbTuiles() const {return tuiles.size();}
32
39     Tuile* piocher(int nbTour, bool riviereActive);
40
45     bool estVide() const {return tuiles.size() == 0;}
46
47     Pioche(const Pioche&)= delete;
48     Pioche& operator=(const Pioche&)= delete;
49     inline bool getBouclier(ContenanceTuile& c) { return c.bouclier; };
50
51
52 };
53
54
55 ostream& operator<<(ostream& f, const Pioche& T);
56
57
58 #endif
```

8.11 Référence du fichier plateau.h

```
#include <vector>
#include <iostream>
#include "tuile.h"
#include "espace.h"
```

Classes

— class Plateau

La classe Plateau permet de gérer le plateau, avec les tuiles, les espaces.

8.12 plateau.h

Aller à la documentation de ce fichier.

```
1 #ifndef PROJET_LO21_CARCASSONNE_PLATEAU_H
2 #define PROJET_LO21_CARCASSONNE_PLATEAU_H
3 #include<vector>
4 #include <iostream>
5 #include "tuile.h"
6 #include "espace.h"
7
16 class Plateau{
```

```
17 private:
18     std::vector<Tuile*> tuiles;
19     std::vector<Espace*> espaces;
20     int nbrEspaces;
21     int nbrTuiles;
22
23 public:
24
30     Plateau(int nbT) : tuiles(nbT), nbrTuiles(nbT){;}
31     Plateau() = default;
32     ~Plateau();
33     int getNbrTuiles() const{return nbrTuiles;}
34     int getNbrEspaces() const{return nbrEspaces;}
35     void setNbrTuiles(const int &nbrT) {this->nbrTuiles = nbrT;}
36
41     void ajouterTuiles(Tuile* tuile);
42
47     void ajouterEspace(Espace* espace);
48
53     void supprimerEspace(Espace* espace);
54     std::vector<Tuile*> getTuiles() const{return tuiles;}
55     std::vector<Espace*> getEspaces() const {return espaces;}
56
63     Tuile* existeTuile(int x,int y){
64         for(Tuile* tuile : tuiles){
65             if(tuile->getX() == x && tuile->getY() == y){
66                 return tuile;
67             }
68         }
69         return NULL;
70     }
71 };
72 #endif //PROJET_LO21_CARCASSONNE_PLATEAU_H
```

8.13 Référence du fichier tuile.h

```
#include <stdio.h>
#include <vector>
#include <string>
#include <iostream>
```

Classes

— class ContenanceTuile
— class Tuile

Espaces de nommage

— namespace std

Énumérations

— enum TypesTuiles {
route , abbaye , ville , champs ,
riviere , auberge , cathedrale , jardin ,
droute }

Fonctions

— ostream & **operator**<< (ostream &f, const Tuile &T)
— ostream & **operator**<< (ostream &f, const ContenanceTuile &T)

8.13.1 Documentation du type de l'énumération

8.13.1.1 TypesTuiles

```
enum TypesTuiles
```

Valeurs énumérées

route	Type 1.
abbaye	Type 2.
ville	Type 3.
champs	Type 4.
riviere	Type 5.
auberge	Type 6.
cathedrale	Type 7.
jardin	Type 8.
droute	Type 9.

8.14 tuile.h

Aller à la documentation de ce fichier.

```
1
2 #ifndef tuile_h
3 #define tuile_h
4
5
6 #include <stdio.h>
7 #include <vector>
8 #include<string>
9 #include<iostream>
10
19 using namespace std;
20
21
22 // Definition enum TypesTuiles
27 enum TypesTuiles {
28     route,
29     abbaye,
30     ville,
31     champs,
32     riviere,
33     auberge,
34     cathedrale,
35     jardin,
36     droute
38 };
39
40
41 // Convertir Enum en String
46 static const char* TypesTuiles_str[] = { "Route", "Abbaye", "Ville", "Champs", "Riviere", "Auberge",
       "Cathedrale", "Jardin", "DoubleRoute" };
47
48
49
50 class ContenanceTuile{
51
52 private :
53
54     friend class Pioche;
55     friend class Tuile;
56
57
```

```
58      int numPlacement;
59      TypesTuiles type;
60      bool bouclier;
61
62  public:
63
70      ContenanceTuile(const TypesTuiles t, const int n): type(t), numPlacement(n), bouclier(false) {}
71      ContenanceTuile() = default;
72
73      inline const TypesTuiles& getType() const {return type;}
74      inline const int getNumPlacement() const {return numPlacement;}
75      const bool getBouclier() const { return bouclier; }
76
80      void setBouclier();
81      void setType(const TypesTuiles t){
82          type = t;
83      }
84
85      inline bool operator==(const ContenanceTuile& c) const {return this->type == c.getType(); };
86
87
88      ContenanceTuile(const ContenanceTuile& c):numPlacement(c.numPlacement), type(c.type),
        bouclier(c.bouclier) {}
89      ContenanceTuile& operator=(const ContenanceTuile& c);
90
91  };
92
93
94
95  class Tuile {
96
97  private:
98
99      Tuile* voisin_haut;
100      Tuile* voisin_bas;
101      Tuile* voisin_gauche;
102      Tuile* voisin_droite;
103      int posX;
104      int posY;
105      vector<ContenanceTuile> contenance;
106
107
108  public :
109
115      Tuile(const vector<ContenanceTuile>& c);
116      Tuile() = default;
117
118      inline const size_t getSize() const {return contenance.size();}
119      inline const int getX() const {return posX;}
120      inline const int getY() const {return posY;}
121      inline const Tuile* getVoisinHaut() const {return voisin_haut;}
122      inline const Tuile* getVoisinBas() const {return voisin_bas;}
123      inline const Tuile* getVoisinGauche() const {return voisin_gauche;}
124      inline const Tuile* getVoisinDroite() const {return voisin_droite;}
125
126      const vector<ContenanceTuile> getContenance() const {return contenance; }
127
133      const ContenanceTuile& getContenance(int i) const {return contenance[i]; }
134      const ContenanceTuile* getContenancePointeur(int i) const {return &contenance[i]; }
135      ContenanceTuile* getContenancePointeur(int i) {return &contenance[i]; }
136
142      const TypesTuiles& getContenu(int i) const { return contenance[i].getType() ; }
143      void changerOrientation();
144
145      Tuile(const Tuile& T);
146      Tuile& operator=(const Tuile& T);
147      bool operator==(const Tuile& T) const{return this->getContenance() == T.getContenance();}
148
149          void setContenu(int i, TypesTuiles c){
150          contenance[i].setType(c);
151      }
156          void ReplaceParChamps(){
157          for (int i = 0; i < 9; i++){
158              if((getContenu(i) == TypesTuiles::auberge) || (getContenu(i)==TypesTuiles::jardin)){
159                  setContenu(i, TypesTuiles::champs);
160              }
161          }
162      }
163
164      void setVoisinHaut(Tuile* t){
165          voisin_haut = t;
166      }
167      void setVoisinBas(Tuile* t){
168          voisin_bas = t;
169      }
170      void setVoisinGauche(Tuile* t){
171          voisin_gauche = t;
```

```
172     }
173     void setVoisinDroite(Tuile* t){
174         voisin_droite = t;
175     }
176     void setPosX(int x){
177         this->posX = x;
178     }
179     void setPosY(int y){
180         posY = y;
181     }
182 
183         //setter x et y
184     void setX(int x){
185         posX = x;
186     }
187     void setY(int y){
188         posY = y;
189     }
190 
191 };
192 
193 
194 ostream& operator«(ostream& f, const Tuile& T);
195 ostream& operator«(ostream& f, const ContenanceTuile& T);
196 
197 
198 #endif /* tuile_h */
199 
```

8.15 vueAccueil.h

```
1 #ifndef VUEACCUEIL_H
2 #define VUEACCUEIL_H
3 
4 #include <QMainWindow>
5 
9 namespace Ui {
10 class VueAccueil;
11 }
12 
16 class VueAccueil : public QMainWindow
17 {
18     Q_OBJECT
19 
20 public:
21 
27     explicit VueAccueil(QWidget *parent = nullptr);
28     ~VueAccueil();
29 
33     void setNbrJoueur();
34 
39     int getNbrJoueur()const{return this->nbJoueur;}
40 private:
41     Ui::VueAccueil *ui;
42     int nbJoueur;
43 
44 private slots:
45 
49         void on_pushButton_clicked();
50 };
51 
52 #endif // VUEACCUEIL_H
```

8.16 vueContenuTuile.h

```
1 #ifndef VUECONTENUTUILE_H
2 #define VUECONTENUTUILE_H
3 
4 #include <QWidget>
5 #include "../modele/tuile.h"
6 
10 namespace Ui {
11 class VueContenuTuile;
12 }
16 class VueContenuTuile : public QWidget
17 {
18     Q_OBJECT
19 
20 private:
```

```
21     Ui::VueContenuTuile *ui;
22     TypesTuiles typeTuile;
23     bool bouclier;
24     int idJoueurMeeple;
25     bool meeple;
26
27
28 public:
29
37     explicit VueContenuTuile(TypesTuiles type, bool bouc, QWidget *parent = nullptr);
38
48     explicit VueContenuTuile(TypesTuiles type, bool bouc,int id, bool meeple, QWidget *parent = nullptr);
49     ~VueContenuTuile();
50
54     void setNomCouleurSansM();
55
60     void setNomCouleurAvecM(const int& id);
61     inline TypesTuiles getTypeTuile() const {return this->typeTuile;}
62     inline bool getBouclier()const {return this->bouclier;}
63     inline bool getMeeple()const {return this->meeple;}
64
65     VueContenuTuile& operator=(const VueContenuTuile& c);
66
67 };
68
69 #endif // VUECONTENUTUILE_H
```

8.17 Référence du fichier vueFormNom.h

Espace dédié a la création de label et zone de texte en fonction du nombre de joueur.

```
#include <QMainWindow>
#include <QLabel>
#include <QLineEdit>
#include <QFormLayout>
#include <vector>
```

Classes

— class VueFormNom

8.17.1 Description détaillée

Espace dédié a la création de label et zone de texte en fonction du nombre de joueur.

8.18 vueFormNom.h

Aller à la documentation de ce fichier.

```
1 #ifndef VUEFORMNOM_H
2 #define VUEFORMNOM_H
3
4 #include <QMainWindow>
5 #include <QLabel>
6 #include <QLineEdit>
7 #include <QFormLayout>
8 #include <vector>
9 using namespace std;
10
11
12
13 namespace Ui {
18 class VueFormNom;
```

```
19 }
20
21 class VueFormNom : public QMainWindow
22 {
23     Q_OBJECT
24
25 public:
26
34     explicit VueFormNom(int& nJoueur, vector<int> listeNumExt,QWidget *parent = nullptr);
35     ~VueFormNom();
36     int getNbrJoueur()const{return this->nbrJoueur;}
37
38
39 private slots:
40
41
42     void on_pushButton_2_clicked();
43
44 private:
45     Ui::VueFormNom *ui;
46     QWidget* m_window;
47     int nbrJoueur;
48     QList<QLineEdit*> m_listLineEdit;
49     QList<QLabel*> m_listLabel;
50     QLabel* L_prenom;
51     QLineEdit* m_prenom;
52     vector<int> listeNumE;
53
54 };
55
56
57 #endif // VUEFORMNOM_H
```

8.19 vueJouerTour.h

```
1 #ifndef VUEJOUERTOUR_H
2 #define VUEJOUERTOUR_H
3
4 #include <QWidget>
5
6 namespace Ui {
7 class VueJouerTour;
8 }
9
10 class VueJouerTour : public QWidget
11 {
12     Q_OBJECT
13
14 public:
15
21     explicit VueJouerTour(QWidget *parent = nullptr);
22     ~VueJouerTour();
23
24 private:
25     Ui::VueJouerTour *ui;
26 };
27
28 #endif // VUEJOUERTOUR_H
```

8.20 vueOuPlacerMeeple.h

```
1 #ifndef VUEOUPLACERMEEPLE_H
2 #define VUEOUPLACERMEEPLE_H
3
4 #include <QDialog>
5 #include "vuePartie.h"
6 #include "modele/tuile.h"
7 #include <QAbstractButton>
8
9
10
11 namespace Ui {
12 class vueOuPlacerMeeple;
13 }
14
15 class vueOuPlacerMeeple : public QDialog
16 {
17     Q_OBJECT
18
```

```
19 public:
20
30     explicit vueOuPlacerMeeple(int l,int c, QWidget *parent = nullptr, VuePartie* part = nullptr, Tuile*
       t = nullptr,Controller *con=nullptr);
31     ~vueOuPlacerMeeple();
32     void setTuile(Tuile&);
33
34 private slots:
35     void on_c0_clicked();
36     void on_c1_clicked();
37     void on_c2_clicked();
38     void on_c3_clicked();
39     void on_c4_clicked();
40     void on_c5_clicked();
41     void on_c6_clicked();
42     void on_c7_clicked();
43     void on_c8_clicked();
44
45 private:
46     Ui::vueOuPlacerMeeple *ui;
47     VuePartie* partie;
48     int Nligne;
49     int Ncol;
50     Tuile* tuile;
51     Controller *controller;
52 };
53
54 #endif // VUEOUPLACERMEEPLE_H
```

8.21 vueOuRetirerMeeple.h

```
1 #ifndef VUEOURETIRERMEEPLE_H
2 #define VUEOURETIRERMEEPLE_H
3
4 #include <QDialog>
5 #include "vuePartie.h"
6 #include "modele/tuile.h"
7 #include <QAbstractButton>
8
9
10
11 namespace Ui {
12 class VueOuRetirerMeeple;
13 }
14
15
16 class VueOuRetirerMeeple : public QDialog
17 {
18     Q_OBJECT
19
20 public:
21
28     explicit VueOuRetirerMeeple(VuePartie* partie=nullptr,QWidget *parent=nullptr);
29     ~VueOuRetirerMeeple();
30
31     int containMeeple(VueTuile& vt);
32     void setTuile(Tuile& tuile,VueTuile& vt);
33
34 private slots:
35
39     void on_OK_clicked();
40
41 private:
42     Ui::VueOuRetirerMeeple *ui;
43     VuePartie* partie;
44 };
45
46 #endif // VUEOURETIRERMEEPLE_H
```

8.22 vuePartie.h

```
1 #ifndef VUEPARTIE_H
2 #define VUEPARTIE_H
3
4 #include <QMainWindow>
5 #include "vueTuile.h"
6 #include "../modele/controller.h"
7
8
```

```
9 namespace Ui {
10 class VuePartie;
11 }
12
16 class VuePartie : public QMainWindow
17 {
18     Q_OBJECT
19
20 public:
21
28     explicit VuePartie(Controller* c, QWidget *parent = nullptr);
29     ~VuePartie();
30
34     void setAffichageScore();
35
39     void setAffichageTuile();
40
44     void setJoueurActu();
45
49     void setPlateau();
50     void piocherCarte();
51
58     void placerMeeple(const int Nligne, const int NCol,Tuile* tuile);
59     void retirerMeeple();
60
67     void placerTuile(const int Nligne,const int NCol,Tuile* tuile);
68
69     void updateVueTuileAddM(int l, int c, int p, Tuile* T);
70     void updateVueTuileRemoveM(int l, int c);
71
75     void partieFinie();
76
77     Tuile *tuilePlace;
78     VueTuile* vueTuilePlace ;
79
80
81 private slots:
82
86     void on_zoomIn_clicked();
87
91     void on_zoomOut_clicked();
92
96     void on_bouttonValiderTuile_clicked();
97
101     void on_rotationTuile_clicked();
102
103
104
105
106     void on_bouttonFinPartie_clicked();
107
108 private:
109     Ui::VuePartie *ui;
110     Controller* controller;
111
112 };
113
114 #endif // VUEPARTIE_H
```

8.23 vuePlacementMeeple.h

```
1 #ifndef VUEPLACEMENTMEEPLE_H
2 #define VUEPLACEMENTMEEPLE_H
3
4 #include "modele/tuile.h"
5 #include <QDialog>
6 #include <QAbstractButton>
7 #include "vuePartie.h"
8 #include "vueOuPlacerMeeple.h"
9
10
11 namespace Ui {
12 class vuePlacementMeeple;
13 }
14
15 class vuePlacementMeeple : public QDialog
16 {
17     Q_OBJECT
18
19 public:
20
30     explicit vuePlacementMeeple(QWidget *parent = nullptr, VuePartie* partie = nullptr, const int l = 0,
       const int c = 0,Tuile* tuile = nullptr,Controller* con=nullptr);
```

```
31     ~vuePlacementMeeple();
32
37     inline int getNligne()const {return Nligne;}
38
43     inline int getNCol()const {return NCol;}
44
45 private slots:
46
47     void on_Non_clicked();
48     void on_Oui_clicked();
49
50 private:
51     Ui::vuePlacementMeeple *ui;
52     VuePartie* partie;
53     const int Nligne;
54     const int NCol;
55     Tuile* tuile;
56     Controller* controller;
57 };
58
59 #endif // VUEPLACEMENTMEEPLE_H
```

8.24 vuePlateau.h

```
1 #ifndef VUEPLATEAU_H
2 #define VUEPLATEAU_H
3
4 #include <QWidget>
5
6 namespace Ui {
7 class VuePlateau;
8 }
9
13 class VuePlateau : public QWidget
14 {
15     Q_OBJECT
16
17 public:
18
24     explicit VuePlateau(QWidget *parent = nullptr);
25
29     void setTable();
30     ~VuePlateau();
31
32
33 private:
34     Ui::VuePlateau *ui;
35 };
36
37 #endif // VUEPLATEAU_H
```

8.25 vueRetirerMeeple.h

```
1 #ifndef VUERETIRERMEEPLE_H
2 #define VUERETIRERMEEPLE_H
3
4 #include <QDialog>
5 #include "vuePartie.h"
6 #include "modele/tuile.h"
7 #include <QAbstractButton>
8 #include "vueOuRetirerMeeple.h"
9
10
11
12 namespace Ui {
13 class VueRetirerMeeple;
14 }
15
16 class VueRetirerMeeple : public QDialog
17 {
18     Q_OBJECT
19
20 public:
21
28     explicit VueRetirerMeeple(VuePartie* partie=nullptr,QWidget *parent=nullptr);
29     ~VueRetirerMeeple();
30
31 private slots:
32     void on_oui_clicked();
```

```
33     void on_non_clicked();
34
35
36 private:
37     Ui::VueRetirerMeeple *ui;
38     VuePartie* partie;
39 };
40
41 #endif // VUERETIRERMEEPLE_H
```

8.26 vueScore1Joueur.h

```
1 #ifndef VUESCORE1JOUEUR_H
2 #define VUESCORE1JOUEUR_H
3
4 #include <QWidget>
5
6 namespace Ui {
7 class VueScore1Joueur;
8 }
9
13 class VueScore1Joueur : public QWidget
14 {
15     Q_OBJECT
16
17 public:
18
26     explicit VueScore1Joueur(std::string nomJ,int id, QWidget *parent = nullptr);
27     ~VueScore1Joueur();
28
29 private:
30     Ui::VueScore1Joueur *ui;
31     std::string nom;
32     int id;
33
34 private slots:
35
39     void on_ajouterPoints_clicked();
40
44     void on_retirerPoints_clicked();
45 };
46
47 #endif // VUESCORE1JOUEUR_H
```

8.27 vueTuile.h

```
1 #ifndef VUETUILE_H
2 #define VUETUILE_H
3
4 #include <QWidget>
5 #include "../modele/tuile.h"
6 #include <vector>
7 #include "vueContenuTuile.h"
8
9
10 namespace Ui {
11 class VueTuile;
12 }
13
14 class VueTuile : public QWidget
15 {
16     Q_OBJECT
17
18 public:
19
26     explicit VueTuile(Tuile* tuile ,QWidget *parent = nullptr);
27
34     explicit VueTuile(Tuile* tuile, int p, QWidget *parent = nullptr);
35     ~VueTuile();
36
41     void setContenuTuile(Tuile& tuile);
42
48     void addMeeple(Tuile* tuile, int p,int id);
49
54     inline vector<VueContenuTuile*>& getVueContenuT() {return tabContenuTuile;}
55
56 private:
57     Ui::VueTuile *ui;
58     vector<VueContenuTuile*> tabContenuTuile;
```

```
59 };
60
61 #endif // VUETUILE_H
```

Index