Rapport C++ Room escape

Juin 2020

GUO Ran

Ici est le lien de notre réalisation pour ce projet :
https://youtu.be/QOWUUFnBoRo

Tableau de matière

Tableau de matière

Introduction 3
Diagramme de class UML 4
Conception de gtkmm (game) 6
    Modele.hpp 7
    Vue.hpp 10
    Controleur.hpp 12
    Main.cpp 14
Conception de mysql (database) 16
    Class Player 17
    Class Mysql 17
    Class WindowsPlayer 19
    Class ChoixPlayer 19
    Main.cpp : 19
Problème, solution et optimisation 21
    Dans ce projet, on a vue des problèmes et on les résolu dans ces aspects : 21
    Dans ce projet, on peut faire optimiser nous program dans ces aspects 22
Conclusion 23

Project room escape: GUO Ran

Introduction

La plupart des langages de programmation possède des bibliothèques de composants graphiques, prêts à l' emploi, pour créer ces interfaces graphiques. Il en existe de nombreuses qui peuvent être utilisées avec C++.

Gtkmm (gtk-- ou gtk minus minus) est l'interface C ++ officielle de la populaire bibliothèque graphique GTK. Gtkmm est un logiciel gratuit distribué sous la licence GNU Lesser General Public License (LGPL).

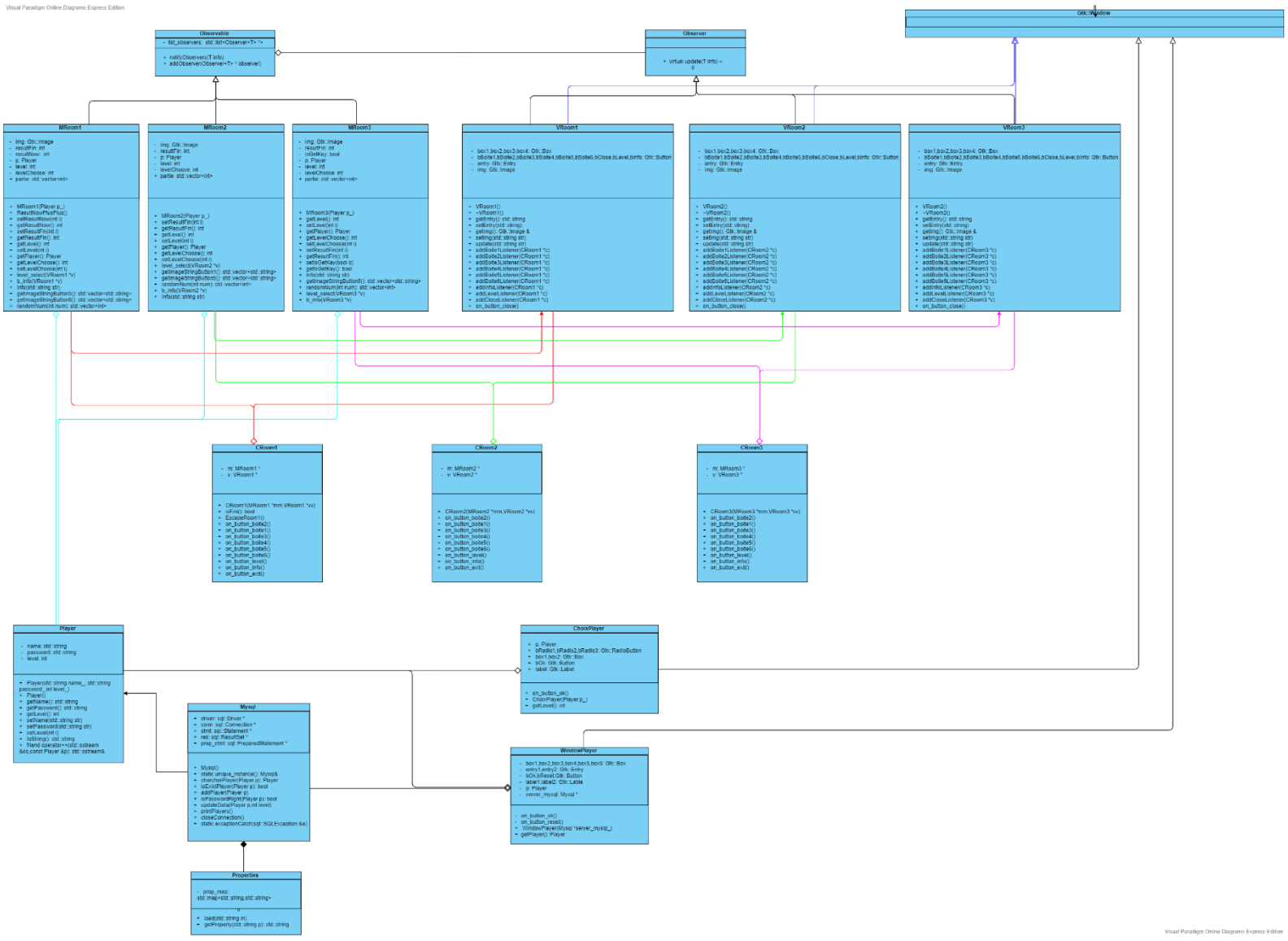
À l' origine, la bibliothèque graphique GTK+ (The GIMP Toolkit)) a été conçue pour le logiciel libre de traitement d' images Gimp, et utilisée, aujourd' hui, en particulier dans l' environnement de bureau GNOME sous Linux

Nous avons l'utilisé pour implémenter un jeu de salle espace

Project room escape: GUO Ran

POLYTECH NICE SOPHIA | UNIVERSITÉ CÔTE D'AZUR

Diagramme de class UML



La figure ci-dessus est la diagramme UML de notre projet C++ : room escape. Car cette figure n' est pas précise, nous vous met ci-joint un fichier sous forme de PDF « UML_C++.pdf».

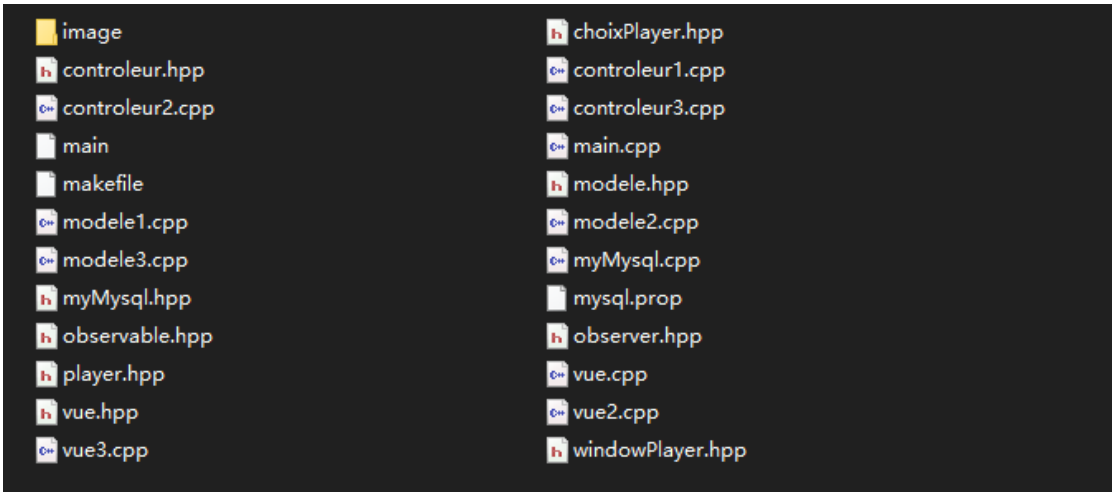
Dans notre projet, il y a 2 parties principales : la partie de jeux en utilisant la librairie de « gtkmm », et la partie de data base grâce à le MySQL.

Dans la partie de jeux, on a 3 chambres, donc on utilise 3 structures de MVC pour réaliser l' interface de graphique. Les méthodes qu' on utilise dans chaque chambre sont écrites

dans les classes MRoom1, MRoom2, MRoom3 dans le « *modèle.hpp* ». Les conceptions d' interface graphique de chaque chambre sont dans les classes VRoom1, VRoom2, VRoom3 dans le « *vue .hpp* ». Les réactions qu' on clique chaque button sont écrites dans les class CRoom1, CRoom2, CRoom3 dans le « *contrôleur .hpp* ».

Dans la partie de data base, nous avons d' abord créé une classe Player qui enregistre le nom, le code secret de ce joueur, et le plus haut niveau auquel le joueur a réussi (Par exemple, ce joueur a réussi à escape de chambre 2, ce niveau est égal à 2.). On a ajouté une classe Mysql pour connecter avec le data base quand il a construit. Il y a 2 interfaces graphiques, WindowPlayer qui nous aide à s' inscrire ou log in, ChoixPlayer qui nous aide à choisir la chambre que on veut jouer. Ces 2 classes tous hérite la classe Gtk ::Window, et ont la relations avec la classe Player et Mysql.

La figure ci-dessous sont tous les fichiers .cpp et .hpp :



Project room escape: GUO Ran

Conception de gtkmm (game)

On réalise cette partie l' interface graphique à l' aide de la librairie gtkmm.

On a créé 3 chambres dans ce projet. Chaque chambre contient 6 petites parties avec la porte. Les joueurs doivent réussir les restes 5 parties pour chercher la clé ou le code secret de la porte, puis ils peuvent sortir de cette chambre.

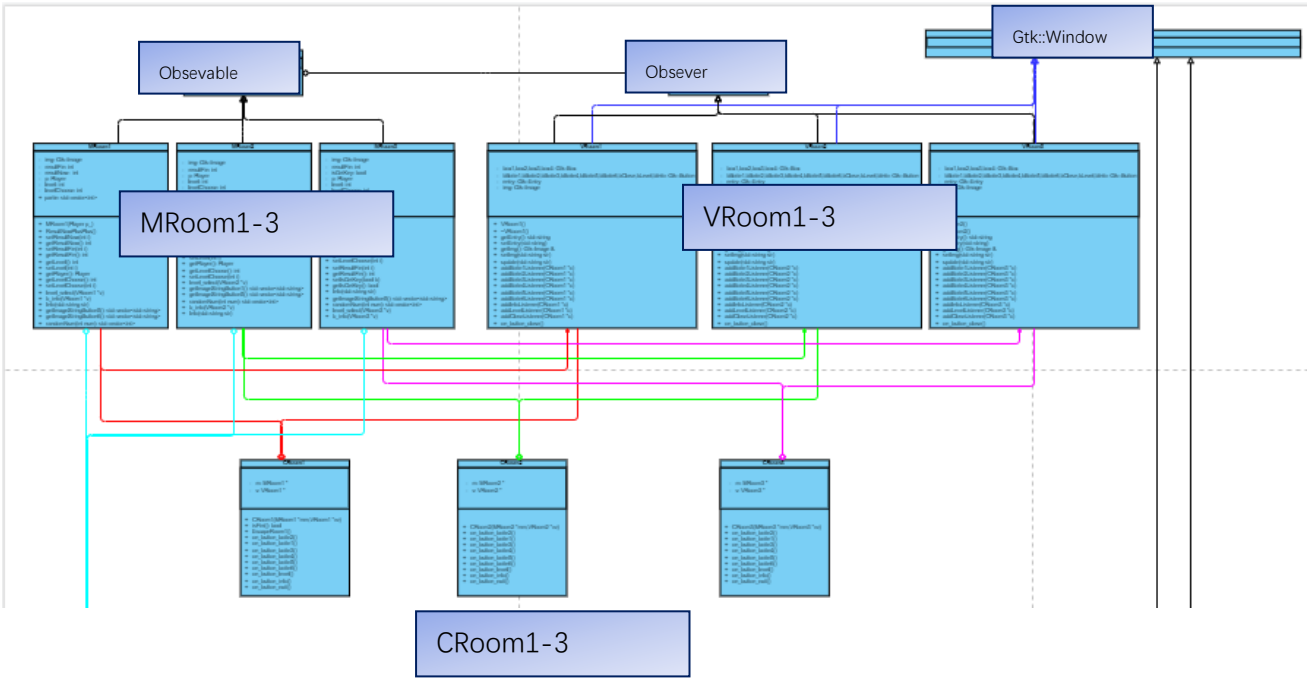
Il y a 3 buttons sous chaque interface :

« Info » peut vous aider à trouver info de joueur ou info de la chambre.

« Level select » peut vous aider à choisir la chambre que vous voulez jouer (changer) ou recommencer cette chambre.

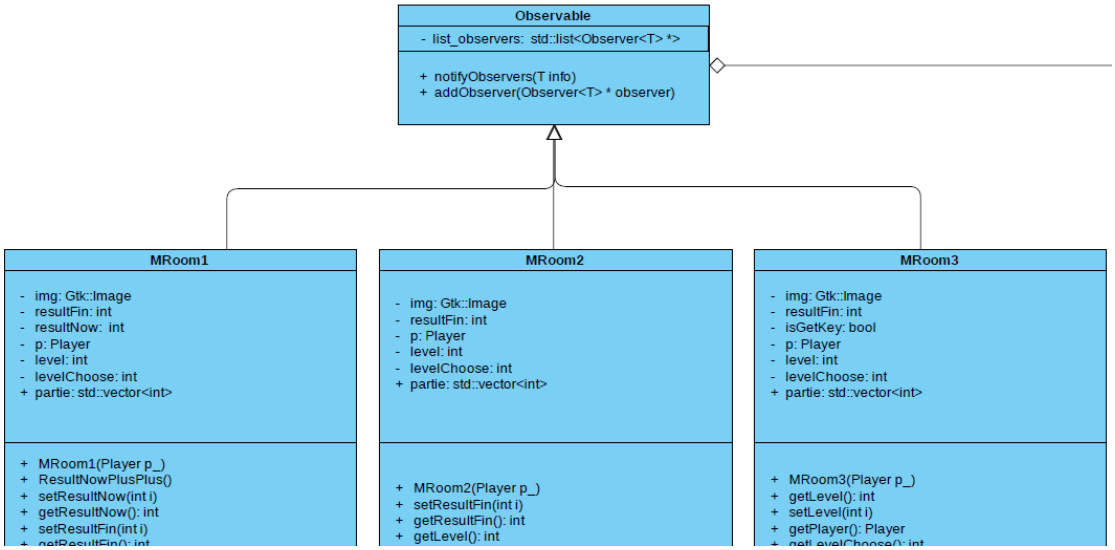
Si vous cliquez le dernier bouton « close », il va finir ce jeu.

Le patron de construction :



Modele.hpp

Dans ce hpp, on a créé 3 classes modèles de chaque chambre MRoom1, MRoom2 et MRoom3 qui héritage d' Observable. Comme la figure ci-dessous :



Le but de la classe Observable :

- Créer une dépendance entre plusieurs objets, les observateurs, et l' état d' un objet particulier, le Modèle.
- Les observateurs s' enregistrent auprès de modèle.
- Quand l' état du modèle change, les observateurs en sont informés, et peuvent se synchroniser.

L' intérêt de la class3 Observable :

- Éviter un couplage fort entre les observateurs et le modèle

Rôle des classes modèle :

Ces classes contient des variables membres et des méthodes pour être appelées par la méthode dans les classes Controleur.

Par exemple dans class MRoom1, on a des variable membres :

```
int resultFin;
int resultNow; // si
Player p;
int level;
int levelChoose;
```

resultFin présent si le joueur a réussi tous les 5 parties de cette chambre, si oui, on met l' un, et on peut ouvrir la porte en ce moment-là, Si non, le joueur n' a pas fini ces parties

et il ne peut pas sortir de cette chambre.

resultNow présent combien de partie a déjà fini par le joueur. Si il est égal à 5, on peut met resultFin=1 qui implique on a réussi cette chambre maintenant.

P est le joueur qui est un type de Player, c' est pour nous update son plus haut record. Et aussi pratique pour nous obtenir les infos sur ce joueur quand on clique le bouton « info » qui est au-dessous d' interface graphique.

Level est le numéro de cette chambre.

LevelChoose est utilisé quand on va changer la chambre et clique le bouton « level select » au-dessous de l' interface graphique. On enregistre le niveau que joueur a choisi et changer la chambre.

Et on a des méthodes dans MRoom1 :

```
void ResultNowPlusPlus(){this->resultNow++;}
void setResultNow(int i){this->resultNow=i;}
int getResultNow(){return this->resultNow;}
void setResultFin(int i){this->resultFin=i;}
int getResultFin(){return this->resultFin;}
int getLevel(){return this->level;}
void setLevel(int i){this->level=i;}
Player getPlayer(){return this->p;}
int getLevelChoose(){return this->levelChoose;}
void setLevelChoose(int i){this->levelChoose=i;}
void level_select(VRoom1 *v);
void b_info(VRoom1 *v);
void Info(std::string str){ notifyObservers(str);}
std::vector<std::string> getImageStringButton5();
std::vector<std::string> getImageStringButton6();
std::vector<int> randomNum(int num);
```

Un constructeur est obligatoire pour initialiser cette chambre.

Des accesseurs et mutateurs des variables privés.

Level_select et b_info est appelé quand on a cliqué le bouton « level select » et « info ».

GetImageStringButton5 et 6 va return les noms des images pour met les images dans interface graphique.

randomNum est pour initialiser le petit jeu quand on clique le bouton aléatoire. Ce là il va changer les images chaque fois.

Ce que on doit faire attention est le méthode Level_select. Selon notre réflexion, ce joueur ne peut jouer que le niveau précédent et le niveau suivant. Donc, si ce joueur est un nouveau joueur, il ne peut pas choisir les deuxième et troisième salle, mais quand il réussit le premier niveau et entre dans la deuxième chambre, il peut choisir la salle précédente (1er chambre).

Pour le réaliser, on doit d' abord juger le plus niveau ce joueur a réussi à l' aide de data bases MySQL et le class Player.

```
case(2):{
  if(this->getPlayer().getLevel()>=1) {
Gtk::MessageDialog dialog2(*v, "Vous etes
```

Comme cette figure, s' il a fini le niveaux 1, il peut choisir la chambre 2. Si non, il y a un message dialogue pour indiquer il ne peut pas choisir cette chambre.

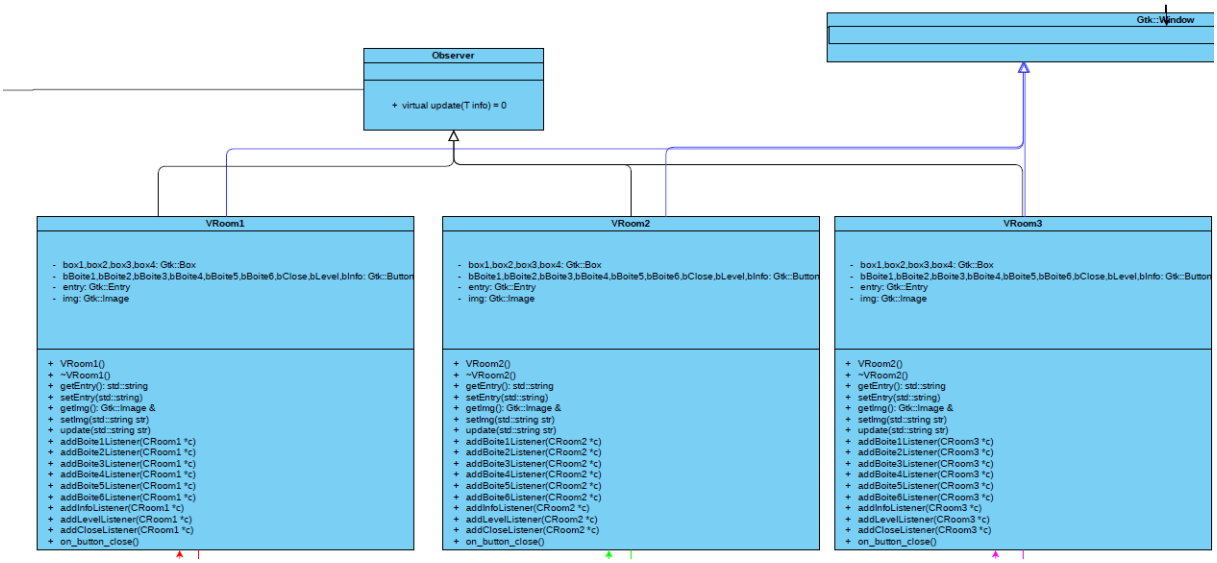
Dans cette méthode, un autre chose on doit faire attention, si le joueur choisi cette chambre qu' il joue maintenant, par exemple, il est dans la chambre 2 et il a choisi le niveau 2, comment on doit faire ? Dans notre programme, le joueur peut choisir continue ou recommencer à l' aide de un message dialogue, comme la figure ci-dessous. S' il choisi OUI, on va recommencer cette chambre-là.

```
case(1):{
  Gtk::MessageDialog dialog2(*v,"Vous etes dans room1 maintenent,voulez vous le recommencer?",
  dialog2.add_button("OUI",1);
  dialog2.add_button("NON",2);
  int result2=dialog2.run();
  if (result2==1){
```

MRoom2 et MRoom3 ont presque même principe que MRoom1.

Vue.hpp

Dans ce hpp, on a créé aussi 3 classes modèles de chaque chambre VRoom1, VRoom2 et VRoom3 qui héritage d' observer et Gtk::Window. Comme la figure ci-dessous :



Rôle des classes vues :

Ces classes ont constitué l' interface graphique de chaque chambre. Et grâce à certaines méthodes pour établir le contact avec le contrôleur, afin d'obtenir la réaction après avoir cliqué sur le bouton.

La classe Observer est un class abstrait qui a une méthode update dont le paramètre

```
template<typename T>
class Observer{
public:
  virtual void update(T info) = 0;
};
```

fournit l' information qu' il doit mettre à jour.

Dans chaque classe de vue, on overrides cette méthode update pour renouveler ce que on vaut, ici on le met pour renouveler info dans l' entry.

```
void VRoom1::update(std::string str) {
  if (str[0]=='I' && str[1]=='n'){
    setEntry(str);
    std::cout<<"je suis update setEntry:"<<str<<std::endl;
  }
  else{
    setImg(str);
    std::cout<<"je suis update setImg:"<<str<<std::endl;
  }
}
```

Et la méthode setEntry est comme ci-dessous :

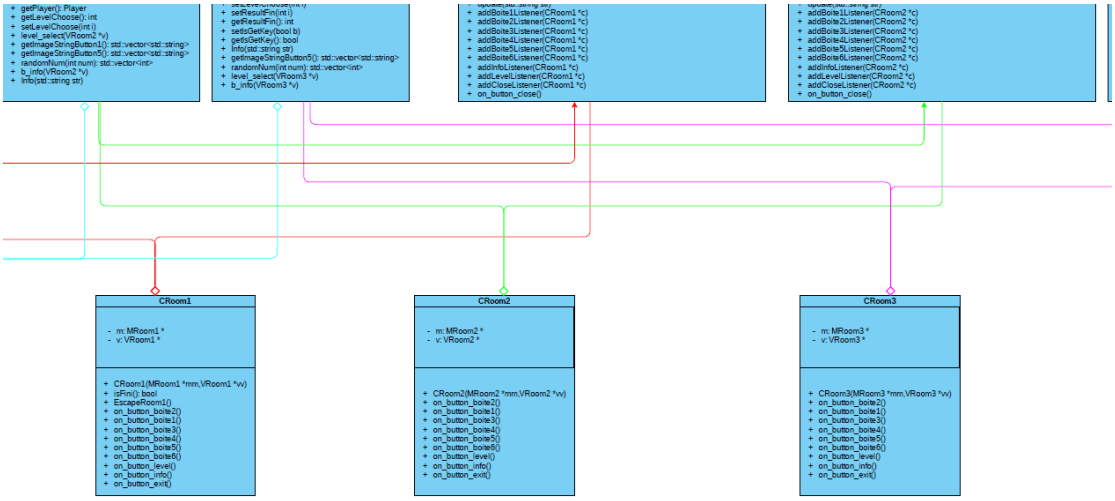
```
void VRoom1::setEntry(std::string str){
  Glib::ustring text =Glib::ustring::format(std::fixed,std::setprecision(2),str);
  entry.set_text(text);
}
```

Dans ces classes, chaque bouton a un relié avec son contrôleur, par exemple le bouton boite2 :

```
void VRoom1::addBoite2Listener(CRoom1 *c){
  bBoite2.signal_clicked().connect(sigc::mem_fun(*c,&CRoom1::on_button_boite2));
}
```

Controleur.hpp

Dans ce hpp, on a créé aussi 3 classes modèles de chaque chambre CRoom1, CRoom2 et CRoom3 qui ont les relations avec les classes modèles et les classes vues. Comme la figure ci-dessous :



Rôle des classes vues :

Le contrôleur qui assure la connexion entre les 2 parties précédentes. On met on_button_clique méthodes ici, quand on clique le bouton, il va appeler ces méthodes pour réaliser ce que on vaut.

```
class CRoom1{
private:
  MRoom1 *m;
  VRoom1 *v;
public:
  CRoom1(MRoom1 *mm,VRoom1 *vv):m(mm),v(vv){
    v->addBoite1Listener(this);
    v->addBoite2Listener(this);
    v->addBoite3Listener(this);
    v->addBoite4Listener(this);
    v->addBoite5Listener(this);
    v->addBoite6Listener(this);
    v->addInfoListener(this);
    v->addLevelListener(this);
    v->addCloseListener(this);
```

```
    void on_button_boite2();
    void on_button_boite1();
    void on_button_boite3();
    void on_button_boite4();
    void on_button_boite5();
    void on_button_boite6();
    void on_button_level();
    void on_button_info();
    void on_button_exit();
};
```

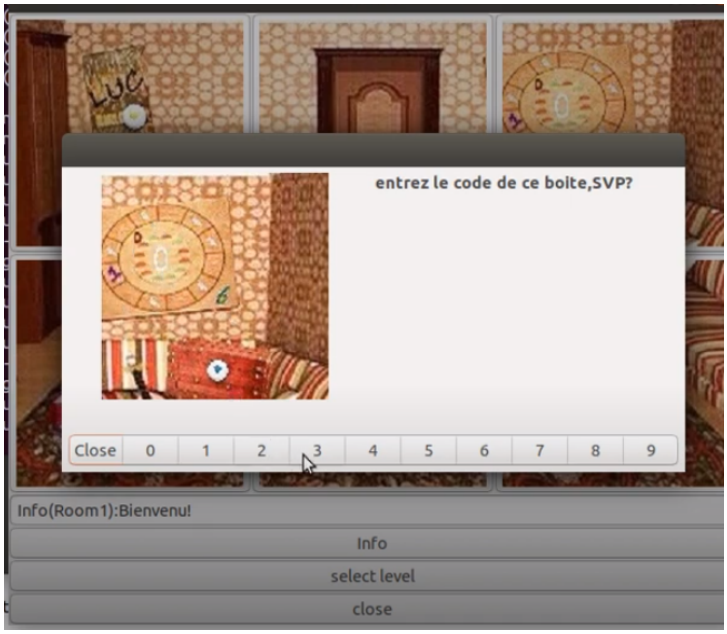
On préfère utiliser le message dialogue, quand on clique un bouton, il va l' ouvrir, et puis on réalise les différentes réactions comme entrer le code secret, un petit jeu de choisir les 2 photos identique etc.

La figure ci-dessous est le code pour réaliser le code secret :

```
m->Info("Info(Room1):Bienvenu!");
Gtk::MessageDialog dialog(*v,"entrez le code de ce boite,SVP?",false,Gtk::MESSAGE_INFO,Gtk::BUTTONS_CLOSE);
dialog.set_image(*imgg);
imgg->show();

dialog.add_button("0",0);
dialog.add_button("1",1);
dialog.add_button("2",2);
dialog.add_button("3",3);
dialog.add_button("4",4);
dialog.add_button("5",5);
```

A la fin l' interface graphique de code secret est :



Et pour le bouton select level et info, on appel les méthodes dans MRoom (modèle)

```
void CRoom1::on_button_info(){
  m->b_info(this->v);
}
void CRoom1::on_button_level(){
  m->level_select(this->v);
  // m->Info("Info(Room1):Bienvenu!");
  }
```

Main.cpp

Main.cpp est principalement pour la progression et la mise en page de l'ensemble du jeu, afin d'entrer dans la salle suivante, de sélectionner une salle ou de redémarrer le jeu.

On a déclaré une variable « level » pour enregistrer laquelle chambre le joueur y reste.

On utilize un switch(level) pour choisir la chamber et entrer dans la chambre suivante.

Ci-dessous est le code switch :

```
int level=pLevel->getLevel();
int result=1;
std::cout<<"Info(User):"<<"this player choose level : "<<level<<std::endl;
while(result==1){
  switch(level){
  case(1):{
    result=0;
     auto app1=Gtk::Application::create(argc,argv);
     MRoom1 *m1=new MRoom1(p);
     VRoom1 *v1=new VRoom1();
     CRoom1 *c1=new CRoom1(m1,v1);
     m1->addObserver(v1);
     app1->run(*v1);
     if (m1->getResultFin()==1){
     server_mysql.updateData(p,1);
     ++level;
     result=1;
     }
     if (m1->getLevelChoose()==1){
level=m1->getLevel();
      result=1;
m1->setLevelChoose(0);
     }
     break;
```

Dans la case rouge, si le joueur a fini cette chambre, on a la variable resultFin==1 qui est dans la classe modèle. En ce cas on met level++ pour entrer dans la chambre suivant, result=1 représente il va continuer ce jeu.

Dans la case verte, si le joueur clique le bouton « select level », on a level=le niveaux joueur a choisi. Result=1 pour continuer ce jeu.

Sinon les 2 cas, result=0, le joueur n' a pas fini cette partie, il n' a pas choisi d' autre chambre et il a cliqué close. En ce cas, result=0, il ne remplis pas la condition du cycle, c' est-à-dire partir de ce jeu.

Ici on a fait attention que si le joueur a fini 3eme chambre, comment on doit faire ? Partir ce jeu ou choisir d' autre nivaux ?

On a mis un message dialogue d' abord dans controleur3.cpp, si le joueur a trouvé le clé de cette chambre, il doit choisir « partir » ou « choisir d' autre level » :

```
    Gtk::MessageDialog dialog(*v,"Info(Room3):Reussir! Vous avez fini tous les level!!! Voulez v
    Gtk::Image *imgg2=new Gtk::Image("image/room3/b2.png");
    dialog.set_image(*imgg2);
    imgg2->show();
dialog.add_button("choisit d'autre level",1);
dialog.add_button("partir",0);
int result2=dialog.run();
if(result2==1){
m->setResultFin(1);
}
else{
   Gtk::MessageDialog dialog3(*v,"Au revoir! Bye!",false,Gtk::MESSAGE_INFO,Gtk::BUTTONS_CLOSE);
   dialog3.run();
}
    m->Info("Info(Room3):Reussir! Vous avez fini tous les level!!");
dialog.hide();
    v->on_button_close();
   }
```

S' il va continuer ce jeu et choisir d' autre level, on met resultFin=1 et on va main.cpp. Sinon, il va partir, on close le window direct.

Dans main.cpp on voit room3, si resultFin=1, on fait level++, en ce moment-là, level=4.

```
  app3->run(*v3);
  if (m3->getResultFin()==1){
    server_mysql.updateData(p,3);
    ++level;
result=1;
 }
  if (m3->getLevelChoose()==1){
 level=m3->getLevel();
     result=1;
 m3->setLevelChoose(0);
   }
  break;
}
default:{
   auto appLevel=Gtk::Application::create(argc,argv);
   Player p2=server_mysql.chercherPlayer(p);
   ChoixPlayer *pLevel=new ChoixPlayer(p2);
   appLevel->run(*pLevel);
   level=pLevel->getLevel();
}
```

On redémarrer le window ChoixPlayer pour choisir la chambre que le joueur vaut jouer.

Conception de mysql (database)

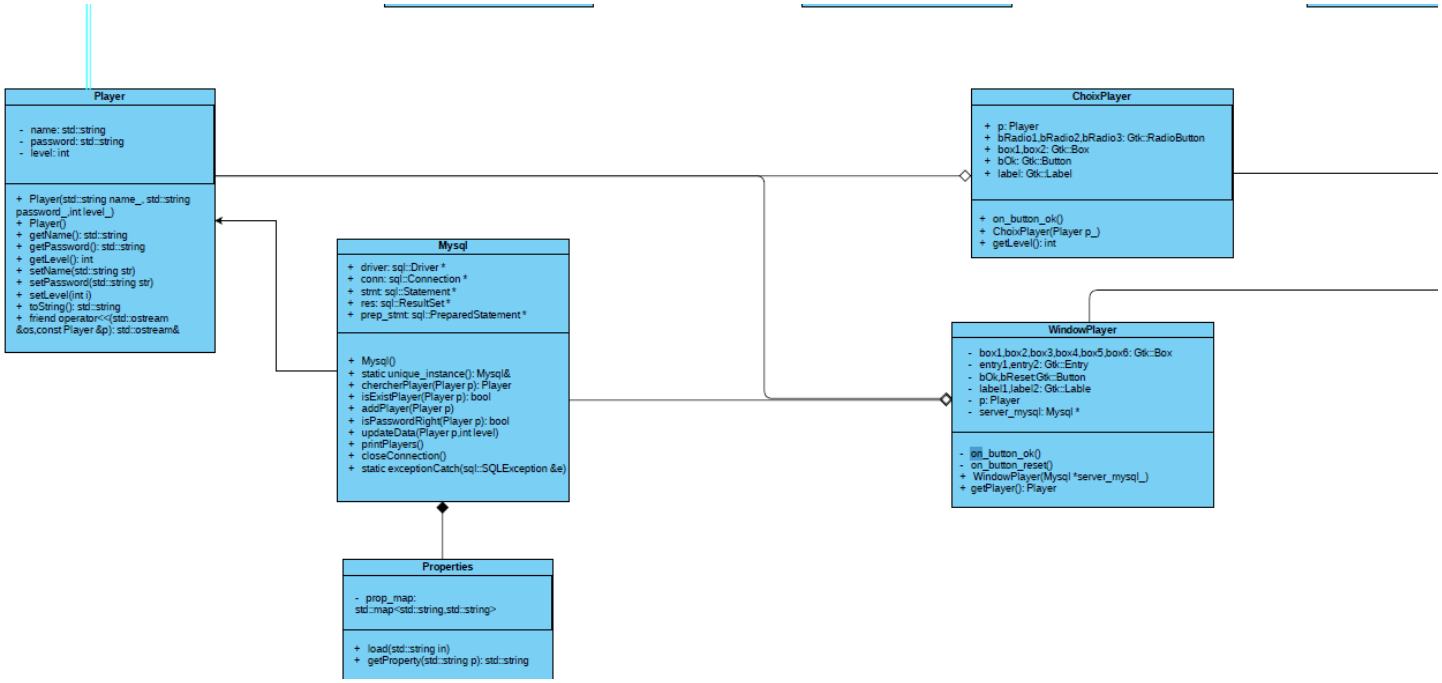
On réalise cette partie data base grâce à MySQL.

On a au total 5 classes dans cette partie :

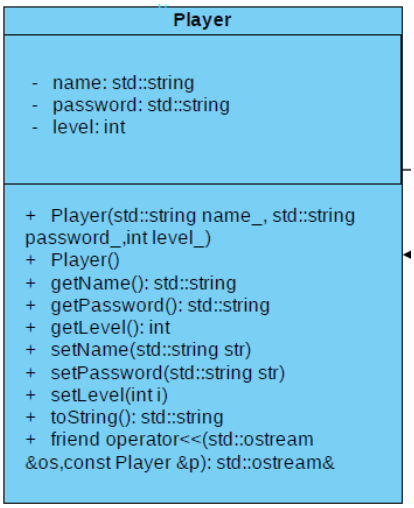
- ✧ Player : pour enregistrer le nom, le code secret, et le plus niveau maintenant de ce joueur.
- ✧ Properties : Pour load le username et password de MySQL dans le fichier « mysql.prop » .
- ✧ Mysql : Pour établir une connexion avec MySQL.
- ✧ WindowsPlayer : héritage de Gtk::window, pour réaliser un interface graphique login et s' inscrire joueur.
- ✧ ChoixPlayer : héritage de Gtk::window, pour réaliser un interface graphique de choisir le niveaux que le joueur vaut jouer.

Rôle : enregistrer les datas des joueurs.

Le patron de construction :

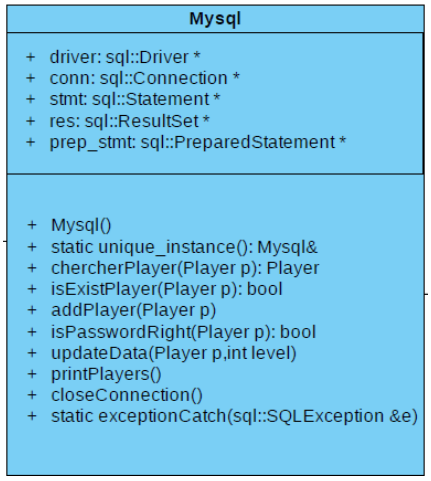


Class Player



Cette class est simple avec des simples méthodes accesseur et mutateur.

Class Mysql



Dans cette classe on a des méthodes de traite des datas dans data bases.

- ✧ isExistPlayer cherche s' il a ce joueur dans data base, on fait un cycle pour chercher le nom de joueur.

```
bool Mysql::isExistPlayer(Player p){
  this->res=this->stmt->executeQuery("SELECT * From players");
  while(this->res->next()){
    if(this->res->getString("name")==p.getName()){
      //std::cout<<"Info(User): Exist player "<<p.getName()<<std::endl;
      return true;
    }
  }
  // std::cout<<"Info(User): Not Exist player "<<p.getName()<<std::endl;
  return false;
```

- addPlayer: ajoute un nouveau joueur dans data base avec son nom, password et level.

```
void Mysql::addPlayer(Player p){
  if(!(this->isExistPlayer(p))){
    this-> prep_stmt->setString(1,p.getName());
    this-> prep_stmt->setString(2,p.getPassword());
    this-> prep_stmt->setInt(3,p.getLevel());
    this-> prep_stmt->execute();
}
```

- cherchePlayer : si ce joueur est existé, on connait son nom, et on peut obtenir son password et le niveau plus haut. On le renvoie sous type Player

```
Player Mysql::chercherPlayer(Player p1){
  Player p("null","null",0);
  if(this->isExistPlayer(p1)) {
    this->res=this->stmt->executeQuery("SELECT * From players WHERE name='"+p1.getName()+"'");
  if(this->res->next()){
    p.setName(this->res->getString("name"));
    p.setPassword(this->res->getString("password"));
    p.setLevel(this->res->getInt("level"));
    }
  return p;
  }
```

- isPasswordRight est pour match le nom et le code. Si le code n' est pas correct, il va renvoyer false.

```
bool Mysql::isPasswordRight(Player p){
  this->res=this->stmt->executeQuery("SELECT * FROM players");
  while(this->res->next()){
    if((this->res->getString("name")==p.getName()) && (this->res->getString("password")==p.getPassword())){
    return true;
      }
    else if((this->res->getString("name")==p.getName()) && (this->res->getString("password")!=p.getPassword())){
      return false;
    }
  }
```

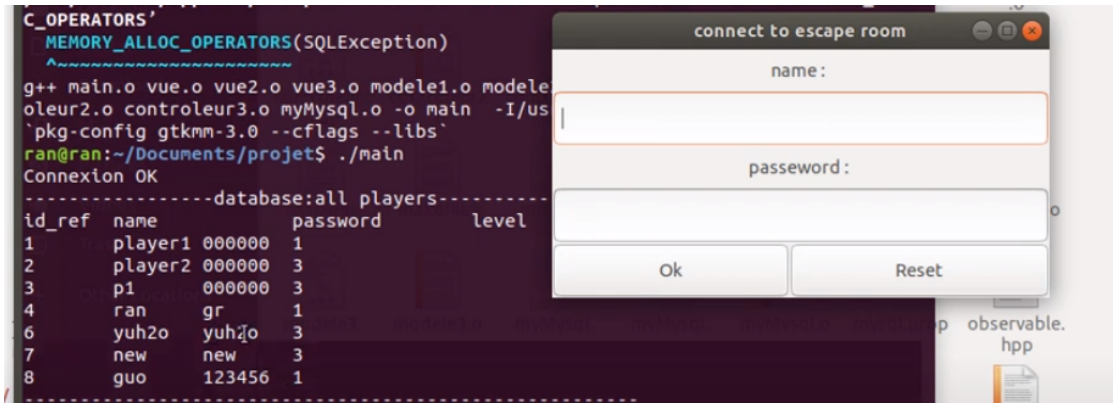
- updateData est pour update le plus haut niveau que ce joueur a réussi. On doit faire attention ici : par exemple, le joueur a déjà réussi la 3eme chambre et puis il a joué la 1ere chambre, en ce moment-là, on ne peut pas renouveler le level comme 1. Donc on doit comparer ce level avec le plus haut niveau que ce joueur a déjà fini.

```
void Mysql::updateData(Player p,int level){
  this->res=this->stmt->executeQuery("Select * from players where name = '"+p.getName()+"'");
  int highLevel=0;
  if(this->res->next()){
    highLevel=this->res->getInt("level");
  }
  if (level>highLevel){
    this->stmt->executeUpdate("Update players SET level='"+std::to_string(level) +"'WHERE name='"+p.getName()+"'");
  }
}
```

- printPlayers est pour afficher tous les datas de joueur sur écran, c' est pratique pour nous voir.

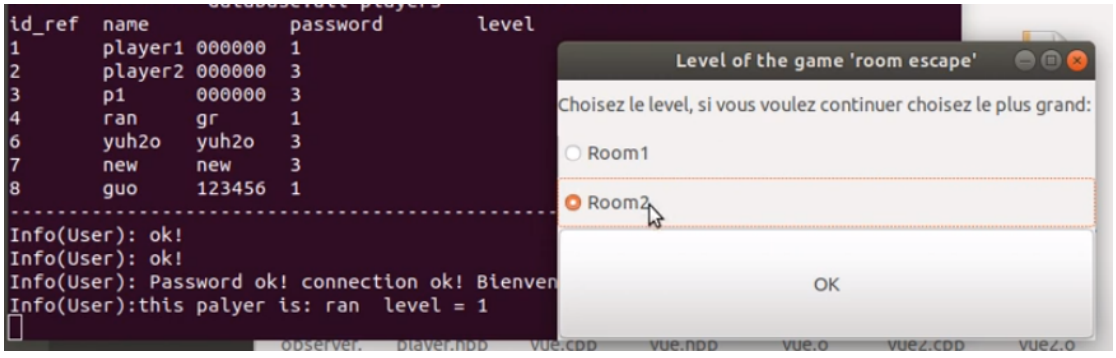
Class WindowsPlayer

Héritage de Gtk::window, pour réaliser un interface graphique login et s' inscrire joueur.
On a l' interface graphique comme la figure ci-dessous :



Class ChoixPlayer

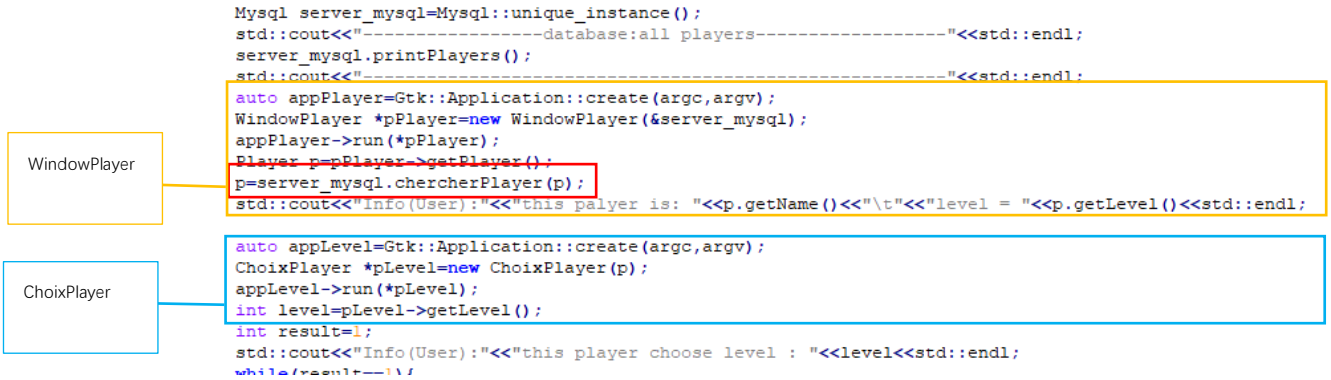
Héritage de Gtk::window, pour réaliser un interface graphique de choisir le niveaux que le joueur vaut jouer.
Dans cette classe, on a utilisé radio bouton.
On a l' interface graphique comme la figure ci-dessous :



Main.cpp :

Dans main.cpp on ajoute 2 graphiques comme les figure ci-dessus avant de démarrer ce jeu.

Ici sont les codes :



On doit faire attention de la case rouge, on doit chercher le joueur dans data bases pour obtenir son niveau le plus haut, et puis à partir de ce niveau, il peut choisir le level il vaut jouer. Le même principe qu' avant, si ce joueur on le niveau le plus haut 1, il peut juste choisir room1 et room2, mais le choix room3 est interdit.

Problème, solution et optimisation

Dans ce projet, on a vue des problèmes et on les résolu dans ces aspects :

- Problème :

  On démarre comme un nouveau joueur, dans data base, son niveau le plus haut est 0. Quand on a fini 1ère chambre et entrer dans 2eme chambre, on voit info de ce joueur, son niveau le plus haut est 0.

  Solution :

  Ce problème est comme on update le data juste à la fin de ce jeu. Ce problème nous misons en garde que on doit update le data bases chaque fois on finit une chambre, et relie à la data base pour renouveler l' info de ce joueur.

- Problème :

  Quand on établir un connection avec la data base, il y a un problème car l' objet n' est pas unique.

  Solution :

  - solution 1 :Pour établir un connection avec MySQL uniquement, on ajoute un attribute statique pour créer une seule instance.

```
static Mysql& unique_instance(){
  static Mysql server_mysql;
  return  server_mysql;
}
```

  - Une autre solution on a réfléchi :

    Utilise un singleton qu' on met constructeur dans privé, cela on ne construit une instance que une fois. (cette solution juste une idée on n' utilise pas, on utilise la solution précédente.)

Dans ce projet, on peut faire optimiser nous program dans ces aspects

- On peut simplifier nos classes MVC à l' aide d' héritage. Car nos classes par exemple MRoom1, MRoom2, MRoom3 sont ressemblé et ont des méthodes pour chaque chambre. Nous pouvons créer une classe super abstrait « MRoom », après chaque classe de MRoomx l' héritage et override ses méthodes.

- Car on a modifié souvent les codes pendant ce projet, il reste quelques codes inutiles que nous n' avons pas supprimé tous, ces codes sont verbeux, nous devons optimiser pour leur supprimer.

Conclusion

Dans ce projet, nous connaissons mieux la bibliothèque gtkmm et la base de données MySQL.

Nous pouvons utiliser habilement certaines fonctions de la bibliothèque gtkmm pour la conception d'interface, et nous pouvons utiliser habilement les fonctions liées à MySQL pour lier la base de données pour le stockage utilisateur.

Il y aura quelques problèmes dans ce processus, mais nous pouvons le résoudre par nos propres efforts.

Merci beaucoup à Monsieur Granet de nous avoir aidés à apprendre le C et C++.

Ce programme n'est pas parfait, il a encore beaucoup de chose à améliorer, nous voulons continuer d’ optimiser notre projet à l'avenir