程序设计实习期末大作业

cheatPOJ

指导教师：张勤健　刘家瑛

小组成员：彭莘尧　牛晨雨　周俊廷

二〇二三 年 七 月

一、项目背景

北京大学是中国著名的高等学府之一，拥有悠久的历史和独特的文化底蕴。其校园内有许多知名景点和建筑，如未名湖、博雅塔等。北京大学对游客开放以来，更是吸引了无数人前来观光和游览。

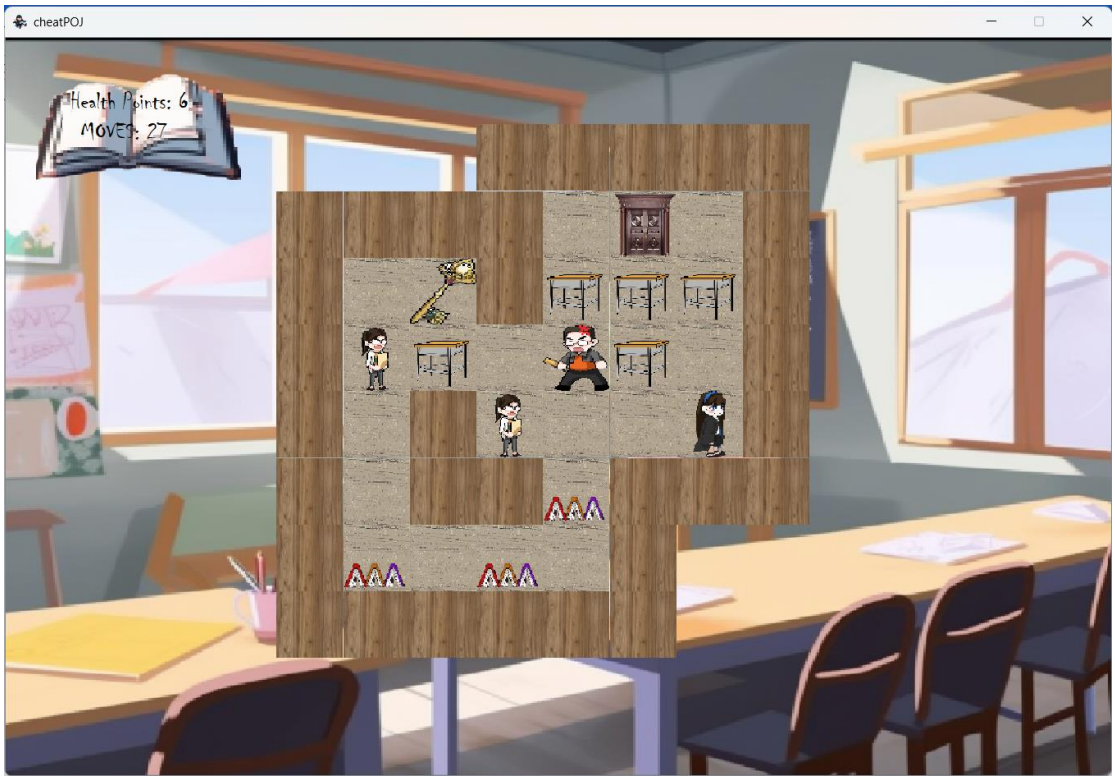
程序设计实习则是北京大学信息科学学院的著名好课，任课的三位老师德高望重，助教们尽职尽责。于是在这门课的大作业的要求发布以来，我们三人为了更好地展示学校的美景和文化遗产，并融合程序设计实习这门课的特点，最终决定开发一款以北京大学校园风景为背景的休闲游戏——cheatPOJ（逃离程设课堂）

二、项目的主要功能

本项目意在实现一款不由米哈游公司开发的“独立非开放世界冒险游戏”。游戏发生一个被称作“程设课堂”的虚拟世界，在这里，被神选中的人将被授予 Qt 的使用权限，引导代码生成。你将扮演课堂上的一名神秘学生，在自由的 coding 中邂逅颜色各异、功能独特的类和其示例，和它们一起修复程序 bug，发掘燕园的秘密——同时逐步发掘“信科教务”的真相。

本游戏原型参考“helltaker”，并借鉴了其中两个关卡的地图作为本游戏的关卡地图。

本游戏的主体是略有改动的推箱子游戏。下面以“stage 2”的场景地图作为示例。



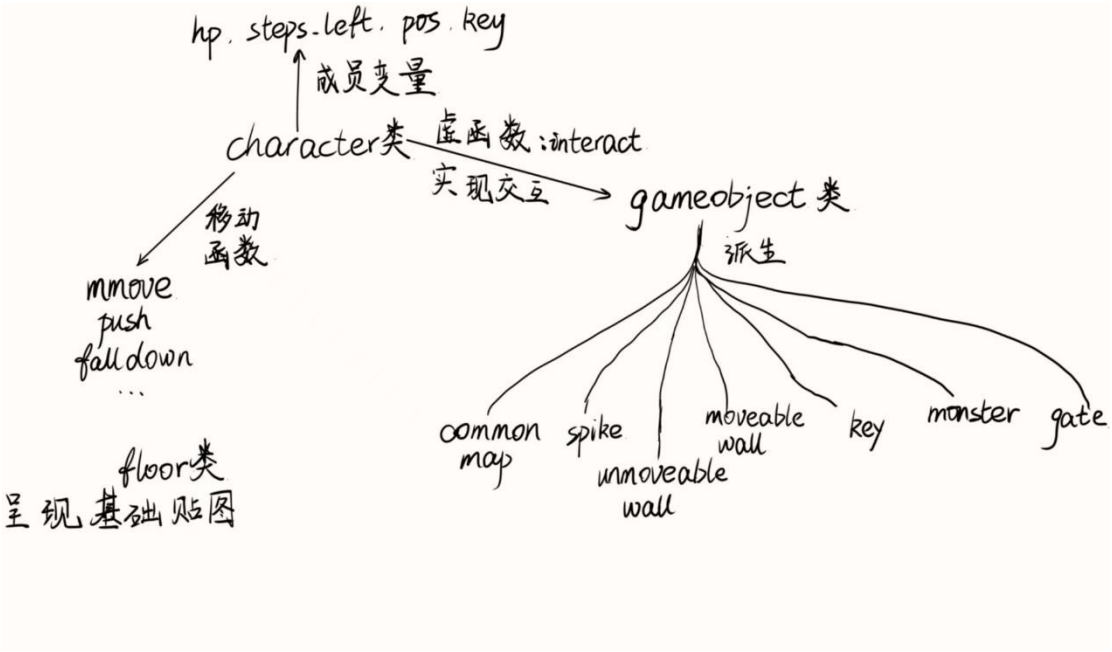
场景地图中有砖块（不可移动的墙体），桌子（可以推动的、类似箱子），五彩斑斓的地刺，教师卡通画（可以吃掉的、类似怪物），精美的钥匙，厚重的门（视为关卡终点），玩家人物贴图。左上角显示的是玩家当前血量（Health Points）和剩余操作数（MOVES）。玩家可以通过上下左右键操控移动，触发特殊动画，Esc 键则可以进入暂停界面。需要注意

的是，吃掉教师卡通画、踩上地刺均会导致生命值减少。同时，未持有钥匙进入门、操作数耗尽、生命值耗尽将分别造成“Presentation Error”“Limit Exceeded”“Wrong Answer”错误，而顺利通关则会显示“Accepted”界面。

游戏提供两个模式供玩家选择：剧情模式和选关模式。进入剧情模式，玩家能够沉浸于我们精心准备的点击式文字冒险游戏，并辅以紧张刺激的推箱子关卡，收获别样游戏体验。进入选关模式，玩家则可以从我们预先设置的四个关卡中任意选择，开启客制化冒险，又是另一番风味。

三、项目代码实现讲解

该项目的组成大致可分为两部分:其一是涉及到后台逻辑的交互用具,以几个类(class)为代表；其二是涉及到前端界面切换的代码，以若干 widget 部件为代表。这里首先从组成这个项目的零部件——类——的设计开始讲解，下面是我们的类的设计图。



其中，floor 类继承自 QGraphicsPixmapItem 类，只是用于生成一个地板的贴图。

我们的 gameobject 类作为一个基类，它的内部有成员变量 posx、posy 用于存储位置，reward 用于设置奖惩效果,type 用于储存此类实例的具体种类(有关具体种类对应的整数，我们将它们作为全局变量定义在了 config.h 中）。由它派生出了 7 个在游戏中不同的，但是都会跟 character 产生交互的物品种类，分别是：

1. common_map：用于占位，没有特殊效果，type=1；
2. spike：刺儿，reward 被设置为-1，type=2；
3. unmoveable_wall：地图边界，不可摧毁，type=3；
4. moveable_wall：呈现为课桌贴图，可以推动，type=4；
5. key：通关必备的钥匙，type=5；
6. monster：呈现为教师卡通贴图，reward 被设置为-1，type=7；
7. gate：关卡的终点，人物的行进目标，type=9.

文件 game.cpp 是我们的游戏运行相关代码，在其中维护了若干二维数组，分别是：

1. groundFloor：用于 floor 类的存储，只是用于显示底层地块贴图。

```
const int ALL_MAPS[4][11][11]
{
    {
        {0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,3,3,3,3,3,3},
        {0,0,0,0,0,3,1,4,8,2,3},
        {0,3,3,3,3,3,2,1,2,3,3},
        {0,3,1,5,2,3,1,1,1,3,0},
        {0,3,4,4,1,3,4,4,4,3,0},
        {0,3,2,1,2,3,1,1,1,3,0},
        {0,3,1,1,1,4,1,1,1,3,0},
        {0,3,1,1,1,4,1,1,1,3,0},
        {0,3,3,3,3,3,3,1,9,3,0},
        {0,0,0,0,0,0,3,3,3,3,0}
    },
    {
```

2. level：是由 int 型组成的 11×11 的二维数组，会在一个游戏类（我们的 game 也是作为一个类来设计的）初始化的时候，根据传入的 level_id，从 config.h 中预编码的地图数组表中找出对应的并复制到这个 level 数组中。我们的预编码地图样式如左：
其中，0 代表空空如也、处于地图外侧的地块，此处在 groundFloor 数组里将使用 nullptr 占位，从而不会产生默认地板贴图；8 代表人物的初始位置；其它数字各自代表我们在类中设计的 gameobject 的对应的 type，我们会在下一步的初始化中使用该数组。

3. level_map：由 gameobject*组成的 11×11 二维数组，根据 level 数组对应生成。通过这个数组，结合 character 自身维护的 posx 和 posy（这里的两个数值就是数组的坐标，其在窗口上的位置能够通过这两个值直接计算得到），我们就能够实现 character 与 gameobject 物体的交互。

接下来是我们的 character 类。如上所述，每当玩家进行一步键盘操作，character 将会计算得到目标位置，并与目标位置通过 character::interact(gameObject*& x)进行交互。这个函数主要做的就是 switch 一下目标 x 的 type 编号，并在此基础上对 character 的状态进行变动。特别地，由于推箱子的步骤需要改变 level_map 中的内容，我们将它放置在了 game.cpp 的主体文件中完成。代码示例如下：

```
1.  if(level_map[tar_row][tar_col]->type==4){
2.      gameObject* checkIfMoveable;
3.      int checkPosRow,checkPosCol;
4.      switch(direction){
5.          case 1: checkIfMoveable = level_map[tar_row][tar_col-
            1]; checkPosRow = tar_row;checkPosCol = tar_col-1;break;
6.          case 2: checkIfMoveable = level_map[tar_row][tar_col+1]; checkPosRow
            = tar_row;checkPosCol = tar_col+1;break;
7.          case 3: checkIfMoveable = level_map[tar_row-
            1][tar_col]; checkPosRow = tar_row-1;checkPosCol = tar_col;break;
8.          case 4: checkIfMoveable = level_map[tar_row+1][tar_col]; checkPosRow
            = tar_row+1;checkPosCol = tar_col;break;
9.      }
10.     if(checkIfMoveable->type==1){
11.         //character push
12.         switch(direction){
13.             case 1: character->push(":/character/push_left.png"); break;
14.             case 2: character->push(":/character/push_right.png"); break;
15.             case 3: character->push(":/character/push_up.png"); break;
16.             case 4: character->push(":/character/push_down.png"); break;
17.         }
18.
19.         delete level_map[checkPosRow][checkPosCol];
20.         delete level_map[tar_row][tar_col];
21.         level_map[checkPosRow][checkPosCol] = new moveable_wall(MOVEABLE_WAL
            L, checkPosRow, checkPosCol, 0, this);
22.         level_map[tar_row][tar_col] = new common_map(COMMON_MAP, tar_row, ta
            r_col, 0, this);
23.
24.         addItem(level_map[checkPosRow][checkPosCol]);
25.         level_map[checkPosRow][checkPosCol]->setPos(200 + CUBE_SIZE*checkPos
            Col, 20 + CUBE_SIZE*checkPosRow);
26.         addItem(level_map[tar_row][tar_col]);
```

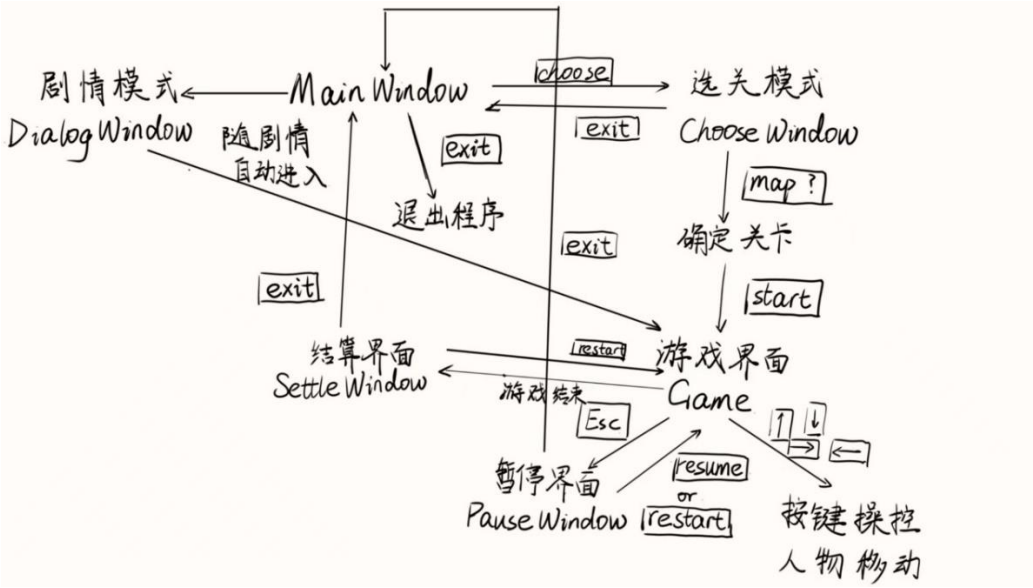
```
27.          level_map[tar_row][tar_col]->setPos(200 + CUBE_SIZE*tar_col, 20 + CU
             BE_SIZE*tar_row);
28.      }else{
29.
30.          switch(direction){
31.              case 1: tar_col+=1;break;
32.              case 2: tar_col-=1;break;
33.              case 3: tar_row+=1;break;
34.              case 4: tar_row-=1;break;
35.          }
36.              //角色跌倒;
37.          character->fallDown();
38.      }
```

以上便是我们 character 类是如何与地图上的 gameobject 类进行交互的讲解。如果 character 只能与物品类进行交互而没有特殊动画展现，这样的游戏无疑是乏味的。于是我们特别根据各个情况，设计了 character 的动画表现，代码示例如下：

```
1.  void Character::fallDown(){
2.      QPixmap pixmap(":/character/fall.png");
3.      QPixmap scaled_pixmap = pixmap.scaled(CUBE_SIZE, CUBE_SIZE);
4.      setPixmap(scaled_pixmap);
5.
6.      change_timer->stop();
7.      change_timer->setInterval(700);
8.      change_timer->start();
9.      connect(change_timer, &QTimer::timeout, this, &Character::stand_up);
10. }
```

这是一个 character falldown 的示例代码。我们是通过改变 character 的 pixmap 实现的，并且通过一个 QTimer 类型的 changetimer 实现动画维持。每个类似的动画会保持 700ms，之后会给 character 发出 standup 信号，从而使得 character 的 pixmap 回到初始站立的形象。我们为 character 设计的图像包括：

以上是有关后台交互的代码讲解。接下来给出我们的前端界面的关系设计图。



再上面的图中，带方框的代表某一个按钮，玩家鼠标点击按钮或键盘敲击按钮可以触发某种界面切换。为了美化我们的界面，是按钮与游戏整体风格吻合，我们选择使用了上面展示过的人物贴图作为按钮贴图。为此，我们定义了一个 FullButton 类继承自 QPushButton 类，代码示例如下：

```
FullButton::FullButton(QString ImagePath, QString Description, QWidget* pare
        nt)
{
    setParent(parent);
    QPixmap pixmap(ImagePath);
    setIcon(pixmap);
    setIconSize(QSize(150, 150));
    setFixedSize(150, 150);
    setStyleSheet("background-color: rgba(255, 255, 255, 5);");
    QFont font("Arial", 13, QFont::Bold);

    QLabel *label = new QLabel(Description);
    label->setParent(this);
    label->setFont(font);
    label->setAlignment(Qt::AlignCenter);
}
```

既然 FullButton 类继承自 QPushButton 类，它便具有后者的全部功能，我们主要使用其中的 clicked 函数判断该按钮是否被点击。同时，FullButton 类有自己的 icon 和 label，让我们能够方便地设计不同种类的按钮，也容易实现按钮选中与否的动画展现。这样的 FullButton 在各个界面中都有使用。

除了按钮使用，玩家也可以通过键盘实现某些操作，这个实现在 mainwindow.cpp 和 game.cpp 中均需要体现。值得一提的是，我们的 MainWindow 界面是继承自 QMainWindow 的，它是一个真正的“window”，而我们定义的其它的各类 window 则是继承自 QWidget 的，相当于只是在 MainWindow 上来来去去改变可见性的若干部件。所以，所有的 QKeyEvent 事实上是由 mainwindow 接受的，因此我们在 mainwindow 里设置函数，将接收到的 QKeyEvent 床给 gamescene，实现在 game 界面上的键盘操作。

这些 window 的设计大同小异，我们以 pausewindow 为例进行展示。

```
//PauseWindow
pausewindow = new PauseWindow();
pausewindow->setParent(this);
pausewindow->hide();
connect(gamescene, &Game::isPaused, this, &pause_now);
connect(pausewindow, &PauseWindow::Resumed, this, &game_resumed);
connect(gamescene, &Game::isResumed, this, &game_resumed);
connect(pausewindow, &PauseWindow::Restarted, this, &game_started);
```

上面的代码是在 mainwindow 里初始化 pausewindow 的操作。setparent 函数是为了让 pausewindow 显示在当前这个窗口上而不会另起炉灶。下面的四个 connect 函数是实现界面交互的核心部件，它们的功能从代码中就能读出来，例如第一行便是将 game 的暂停信号与 mainwindow 的一个 pause_now 函数连接，起到调出暂停窗口的效果。

```
void PauseWindow::init(int _level_id)
{
    int level_id = _level_id;
    setFixedSize(WIDTH, HEIGHT);

    resume = new FullButton(":/button/start.png", "RESUME", this);
    resume->move(350, 400);
    connect(resume, &FullButton::clicked, this, [=]{
        emit Resumed();
          });

          to_exit = new FullButton(":/button/exit.png", "MAIN MENU", this);
```

```
            to_exit->move(640, 400);
            connect(to_exit, &FullButton::clicked, this, [=]{
                emit BacktoMain();
            });

            restart = new FullButton(":/button/start.png", "RESTART", this);

            restart->move(480, 200);
            connect(restart, &FullButton::clicked, this, [=]{
                emit Restarted(level_id);
            });
        }
```

上面的代码是 pausewindow 的 init 方法代码。我们并没有在构造函数中进行操作，这是因为不同的关卡的 pausewindow 应当存储不同的 level_id，这对于 restart（重新游玩关卡）有重要作用。pausewindow 上设置有三个 FullButton：resume，to_exit，restart，分别有 connect 函数作用。接下来再以 pause_now 函数为例：

```
void pause_now(int level_id)
{
    if (interface_mode == _gaming_window){
        pausewindow->init(level_id);
        pausewindow->show();
        blur->setBlurRadius(5);
        gamescene->isgoing = false;
        music->turnDownMusic();
        interface_mode = _pausing_window;

        }
    }
```

这里就使用到了 pausewindow 的 init 方法，init 之后 show。同时为了界面美观，我们设置了一个 blur（模糊）效果，这就是之前函数连接的 pause_now 函数所做的事情。其它的函数类似，其它的界面也是类似的，在此不再赘述。

音乐方面，我们选用了广为人知、脍炙人口的《Super Mario》和《团子大家族》，这一方面符合了作者的喜好，另一方面旋律优美动听，令人心旷神怡。

这里通过创建一个 MusicLoader 类来实现音乐播放、切换等功能。

```
class MusicLoader
{
public:
    MusicLoader();
    void loadMusic(QString music_filename);//加载音乐
    void changeMusic(QString music_filename); //切歌
    void turnDownMusic();    //关闭音乐
    void setSound(int x);  //设置音量
    float curVolume();//当前音量
        bool isPlaying();
        static int sound;   //统一声音大小

    private:
        QSoundEffect *startsound;
    };
```

下面的代码分别实现了加载音乐、切歌、关闭音乐、调节音量的功能，具体方法就是依逻辑调用 QEffectSound 的相关方法。

```
void MusicLoader::loadMusic(QString music_filename){

    startsound->setSource(QUrl::fromLocalFile(music_filename));
    startsound->setVolume(100);
    startsound->setLoopCount(QSoundEffect::Infinite);//设置循环次数
       int;   QSoundEffect::Infinite 枚举值 无限循环
    startsound->play();//软件启动自动播放
```

```
7.
8. }
9.
10.       void MusicLoader::changeMusic(QString music_filename){
11.           startsound->stop();
12.           startsound->setSource(QUrl::fromLocalFile(music_filename));
13.           startsound->setLoopCount(QSoundEffect::Infinite);//设置循环次数
          int；   QSoundEffect::Infinite 枚举值 无限循环
14.           startsound->play();
15.       }
16.
17.       void MusicLoader::turnDownMusic(){
18.           startsound->stop();
19.       }
20.
21.       void MusicLoader::setSound(int x){
22.           startsound->setVolume(x);
23.           sound = x;
24.       }
```

文本显示部分，我们先创建了一个 txtshow 类，在其中实现了一个字符串逐字输出的功能。

```
1. class TxtShow:public QObject
2. {
3.
4. private:
5.     QString str;
6.     QLabel *lab;
7.     QFont font;
8.     const QString words_style_options[6]={"Microsoft YaHei","KaTi","FangSong
          "};  //可选字体
9.     const int words_size_options[6]={10,20,30,40,5,16};//可选字号
10.           int inc;
11.           int idx;
12.           int inter;
13.           int word_size;          //0,1,2,3,4,5
14.           int word_style;         //0,1,2,3,4,5
15.
16.       public:
17.           TxtShow()
18.               :inc(0),word_size(20),word_style(0),inter(100)
19.           {
20.           };
21.
22.           void load(QString st,QLabel *lb);
23.           void setSpeed(int speed); //设置打字速度
24.           void reset();
25.           QString Str();
26.           QString LabStr();
27.           void timChanged();
28.           void setSize(int x);
29.           void setStyle(int index);
30.           //void keyPressEvent(QKeyEvent* event);
31.           void mousePressEvent(QMouseEvent *ev);
32.
33.       };
```

其中 timChange 为实现主要功能的函数，原理为设置时间间隔 inter，用 QTimer 进行暂停，每次在 QLabel 上每次多显示一个字。从而达到了逐字输出的效果。具体代码如下：

```
1. void TxtShow::timChanged()
2. {
3.     setSize(word_size);
4.     setStyle(word_style);
5.     lab->setFont(font);
6.
7.     while(inc<str.size()){
```

```
        if(inc>=str.size()){
            break;
               }
               else
               {
                   lab->setText(str.left(inc));

                   inc++ ;
                   lab->show();
               }

               QEventLoop eventloop;
               qDebug()<<inter;
               QTimer::singleShot(inter, &eventloop, SLOT(quit()));
               eventloop.exec();
           }
           inc=0;
       }
```

此外，还有文本变速功能：

```
void TxtShow::setSpeed(int speed)
{
    inter=speed;
}
```

这将在之后用到。

仅此一个类显然还无法实现完整功能，我们又创建了一个 dialogwindow 类用于显示对话，具体原理与 pausewindow、chooswindow 等相同，便不再在此赘述，而是仅仅展示一下它的接口并调出差异点细述。

```
class DialogWindow:public QWidget
{
    Q_OBJECT
public:
    int order;
    QLabel *background_label;
    FullButton *choice1;
    FullButton *choice2;
    explicit DialogWindow(QWidget *parent = nullptr);
           void startDialog();
           void BacktoGame();
           //void keyPressEvent(QKeyEvent* event);
           void mousePressEvent(QMouseEvent *ev);
       };
```

一位先哲曾言，“没有快进和跳过的剧情都是蓝剧情”，所以我们在这里重载了 mousePressEvent，在其中调用了 setSpeed 方法改变了 QTimer 的间隙大小，用以实现鼠标左键剧情加速的功能。

```
void DialogWindow::mousePressEvent(QMouseEvent *ev)
{
    if(ev->button() == Qt::LeftButton){

        posi = (posi+1)%2;
        txt.setSpeed(speeds[posi]);
        qDebug()<<1;
    }
}
```

对话则是从资源文件中的 txt 文件中读取，我们设置了五段剧情，分别置于五个 txt 文档中；再在 startDialog 中进行读取文件的操作，再调用上文提及的 timChanged 方法，实现整个文本的输出。

四、特别致谢

感谢中央美术学院的许烟溪同学，她为本项目最终呈现的封面、人物、背景等提供了专业的设计及制作支持。