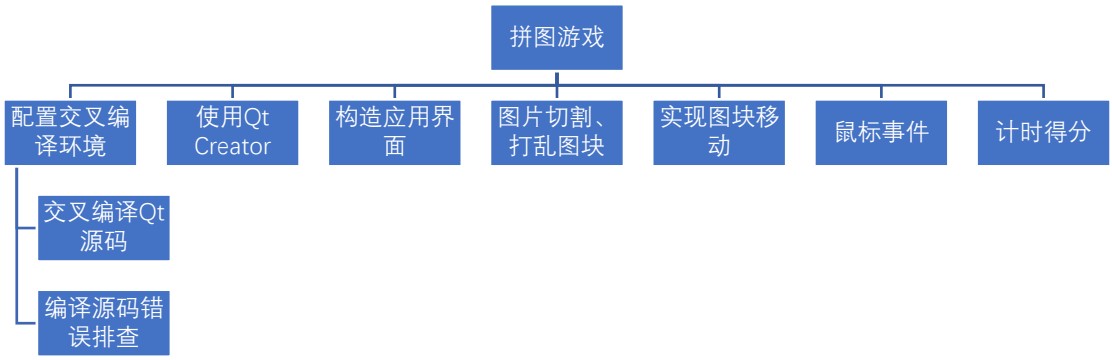
# 第一章　使用龙芯派制作拼图游戏

## 1.1 章节描述

拼图游戏是一种非常古老的游戏，在公元前一世纪就出现了名为“七巧板”的拼图游戏。我们在日常生活中也接触过网页上的拼图游戏，将一幅图片等分后打乱顺序，并挖出一块图块作为拼图活动空间，通过对图块的挪动恢复原状，实在是趣味无穷。

本次设计中，将要求对选择的任意图片切分后打乱顺序，由鼠标拖动错乱的图块将图片恢复原状。在进阶设计中，我们将加入键盘的控制，除了鼠标拖动之外，再增加方向键移动图块的功能。

本章在开发该应用需要了解和掌握的知识点如图所示。



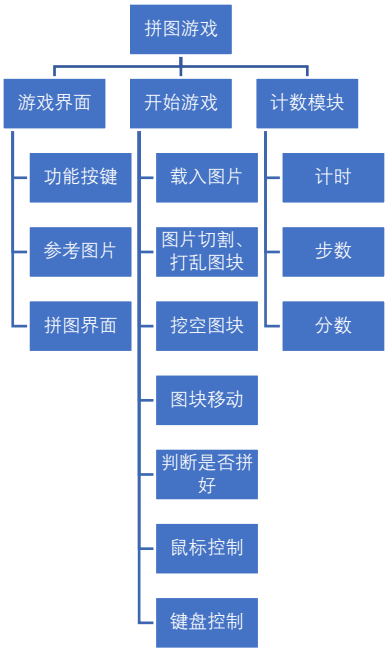

## 1.2 应用设计定义

一个好的设计是成功的一半，在开始写代码之前，不妨放下手中的事情，在一张白纸上把我们要做什么，应用需要的功能，功能之间的切换方式列出来。如果时间充裕，最好能把主要的用户界面简单画一下。最后，再做一个长远的规划，在应用中还要进行哪些扩展，是否要将接口留出。

事实上，这些工作在复杂应用中是需要产品经理配合完成应用定义的，不过这次我们做的应用比较简单，自己就能够完成规划的工作。

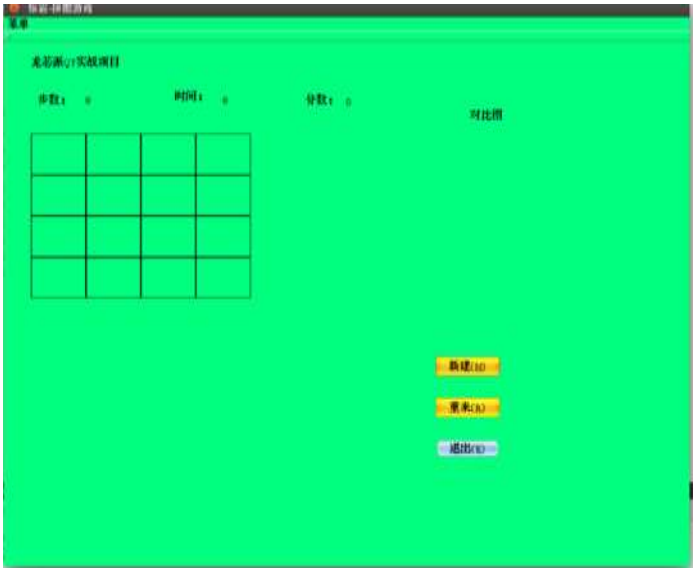
### 1.2.1 应用功能

拼图游戏的功能主要由三个部分组成：游戏界面、游戏交互和计分模块。我们在代码实现的过程中，也将按照下图的功能分步实现：

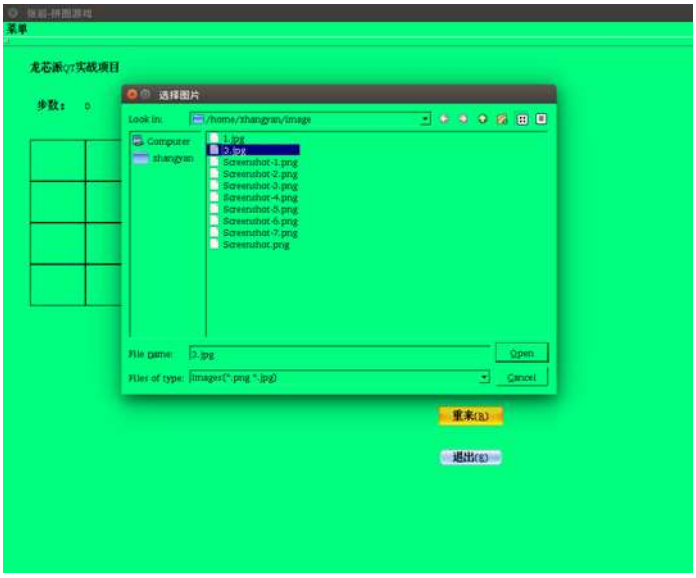



### 1.2.2 应用界面预览

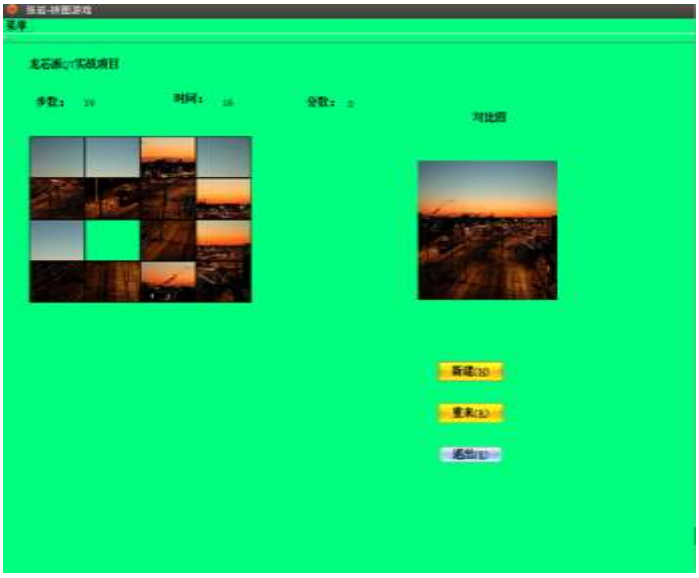
拼图游戏直接使用了 Qt Creator 的默认纯色背景，上方是计分部分，左侧是拼图区，右侧是原图展示区。在拼图完成后，会弹出对话框，显示游戏得分。



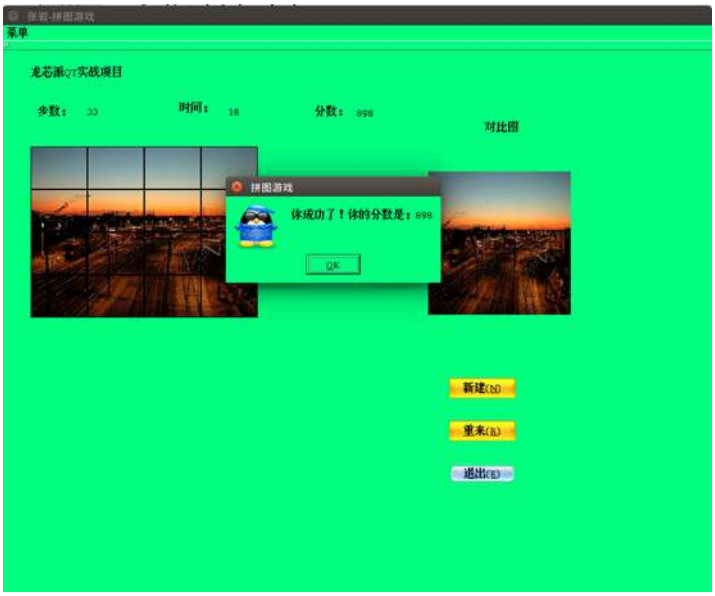
初始界面


菜单
龙芯派QT实战项目
步数：
选择图片
Look in:
/home/zhangyan/image
Computer
zhangyan
1.jpg
2.jpg
Screenshot-1.png
Screenshot-2.png
Screenshot-3.png
Screenshot-4.png
Screenshot-5.png
Screenshot-6.png
Screenshot-7.png
Screenshot.png
File name:
3.jpg
Open
Files of type:
Images(*.png *.jpg)
Cancel
重来(R)
退出(E)

选取图片


菜单
龙芯派QT实战项目
步数：
时间：
分数：
对比图
新建(N)
重来(R)
退出(E)

开始拼图


菜单
龙芯派QT实战项目
步数：
33
时间：
18
分数：
898
对比图
拼图游戏
你成功了！你的分数是：898
OK
新建(N)
重来(R)
退出(E)

拼图成功，返回分数

退出游戏

## 1.3 配置 QT 开发环境

### 1.3.1 开发环境要求

该应用的开发环境要求如下：

- ✓ 上位机操作系统：Ubuntu 16.04
- ✓ 龙芯派操作系统：Loongnix
- ✓ 开发工具：Qt 4.8.6、Qt Creator 4.8
- ✓ 开发语言：C 语言、C++

### 1.3.2 交叉编译的流程

龙芯派作为嵌入式开发设备，在配置 Qt 开发环境时，需要通过龙芯派的交叉编译工具链，将 Qt 的源代码在上位机编译成可执行文件。之后在 Qt Creator 中才能进行龙芯派开发环境的配置。具体的流程如下图所示：


### 1.3.3 下载安装龙芯派的交叉编译工具链

在安装 Ubuntu 的上位机下访问 http://ftp.loongnix.org/embed/ls3a/toolchain/gcc-4.9.3-64-gnu.tar.gz，下载编译需要用到的工具链。

下载后打开命令行终端（Ctrl+Alt+t），在命令行执行以下命令解压该文件。

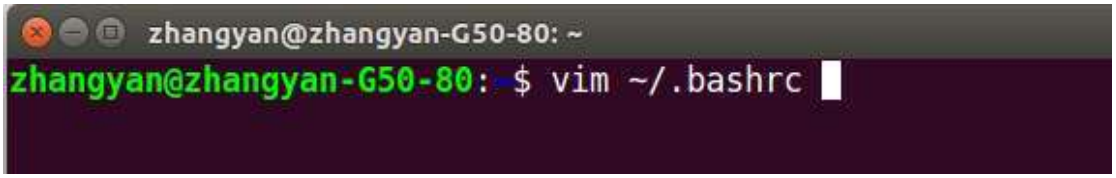
```
tar -xvf gcc-4.9.3-64-gnu.tar.gz
mv  gcc-4.9.3-64-gnu  /opt
```

解压后可以看到，整个工具链是由很多功能文件包组成的，我们只需要使用其中的部分工具链，因此需要通过环境变量的方式指定文件的路径。

环境变量：环境变量（environment variables）一般是指在操作系统中用来指定操作系统运行环境的一些参数，如：临时文件夹位置和系统文件夹位置等。


将解压好的工具链文件 gcc-4.9.3-64-gnu 添加到环境变量中，才能在之后编译 Qt 源码时，对工具链的调用位置准确无误。

（1） 打开家目录（HOME）下的.bashrc




（2） 将 export PATH=$PATH:/opt/gcc-4.9.3-64-gnu/bin 添加到最后一行。一般来说，在终端中执行的路径信息是不会保存的，因此需要通过执行该语句保证下次启动时仍然保存了环境变量。

（3） 然后执行 source ~/.bashrc，使用该语句更新环境配置。

（4） 如果需要确认版本信息，可以执行 mips64el-linux-gcc -v 语句


### 1.3.4 下载 Qt 源码

本项目使用 Qt 配合 Qt Creator 进行开发。Qt 是一个跨平台的 C++图形用户界面应用程序框架。它提供给应用程序开发者建立 GUI（图形用户界面）所需的所用功能。Qt 是完全面向对象的，很容易扩展，并且允许组件编程。

在 Qt 的官网网址： http://download.qt.io/archive/qt/4.8/4.8.6/qt-everywhere-opensource-src-4.8.6.tar.gz 下载 Qt 4.8.6 的源码包后，在命令行输入

sudo tar -xvf qt-everywhere-opensource-src-4.8.6.tar.gz

完成源码包解压。

进入 Qt 的源码目录下，我们可以看到 Qt 的源码目录。


因为当前的 Qt 源码是一个通用版本，是不可以直接在龙芯平台使用的，需要对 qmake 文件进行修改。

qmake 文件：qmake 是用来为不同的平台和编译器书写 Makefile 的工具，是 Qt 库和 Qt 所提供工具的主要联编工具。

在 Qt 源码目录下，需要先修改文件 mkspecs/qws/linux-mips-g++/qmake.conf，，如图所示。

修改前


修改后


### 1.3.5 选择需要的选项参数

Qt 源码中包含通用的组件，部分组件是我们的应用中不需要的，因此需要在选项参数中进行增减。所有选项参数可以在终端执行 sudo ./configure –help 查看。

我们选择了以下几个选项参数：

| 参数 | 功能说明 |
| --- | --- |
| -prefix /opt/Qt4.8mips | 指定想要安装到的安装目录 |
| -opensource | 以开源版本发布程序 |
| -embedded mips | 指定嵌入式平台的架构 |
| -xplatform qws/linux-mips-g++ | 指定目标平台使用的交叉编译工具 |
| -no-webkit | 禁用 webkit 模块 |
| -qt-libtiff | 支持 tiff 插件 |
| -qt-libmng | 支持 mng 插件 |
| -no-mouse-tslib | 不加载触摸驱动 |
| -qt-mouse-pc | 加载鼠标驱动 |
| -no-neon | 不支持 ARM 扩展指令集 NEON |
| -little-endian | 小端存储模式 |
| -shared | 动态编译库 |
| -qt-libpng | 支持 png 图片格式 |
| -qt-libjpeg | 支持 jpeg 图片格式 |
| -qt-kbd-tty | 支持串口控制 |
| -qt-gfx-linuxfb | 加载显示设备 |
| -system-sqlite | 启用支持 sqlite 数据库 |

选定选项参数后，在终端中的 qt 源码顶层目录执行命令 sudo ./configure -prefix /opt/Qt4.8mips -opensource -embedded mips -xplatform qws/linux-mips-g++ -no-webkit -qt-libtiff -qt-libmng -no-mouse-tslib -qt-mouse-pc -no-neon -little-endian -shared -qt-libpng -qt-libjpeg -qt-kbd-tty -qt-gfx-lin uxfb -system-sqlite 完成对选项参数的修改。

在调试中，如果出现报错重启的情况，我们之前执行的命令是不会保存的。可以把修改选项参数的命令做成一个脚本，如果需要重新配置，只需要执行脚本就可以了。

```
zhangyan@zhangyan-G50-80:~/download/test/qt-everywhere-opensource-src-4.8.6$ ls
bin             config.status  demos     include             LICENSE.FDL   mkspecs       README     translations
changes-4.8.6   config.tests   doc       INSTALL             LICENSE.GPL3  plugins       src        util
cmd.sh          configure      examples  LGPL_EXCEPTION.txt  LICENSE.LGPL  projects.pro  templates
config.profiles configure.exe  imports   lib                 Makefile      qmake         tools
zhangyan@zhangyan-G50-80:~/download/test/qt-everywhere-opensource-src-4.8.6$
```

图中的 cmd.sh 就是我们制作的脚本，打开后如下图所示：

```
1 #!/bin/bash
2
3
4 sudo ./configure -prefix /opt/Qt4.8mips-new -opensource -embedded mips -xplatform qws/linux-mips-g++ -no-webkit -qt-libtiff -qt-libmng
    -no-mouse-tslib -no-mouse-linuxtp -no-neon -little-endian -shared -qt-libpng -qt-libjpeg -qt-kbd-tty -qt-gfx-linuxfb
```

### 1.3.6 编译 Qt 源码

在前面的工作准备完成后，来到了最神奇的时刻，我们可以开始神圣的编译环节了，编译环节可能会非常简单，只需要在命令行执行命令

make

等待比较长的一段时间，就能完成编译。在这期间，可以吃一碗泡面或者打一会毛衣。在命令执行完毕，重新会到命令行提示符下时，如果没有出现 error 等字样，那么就编译完成了。

随后后输入命令

make install

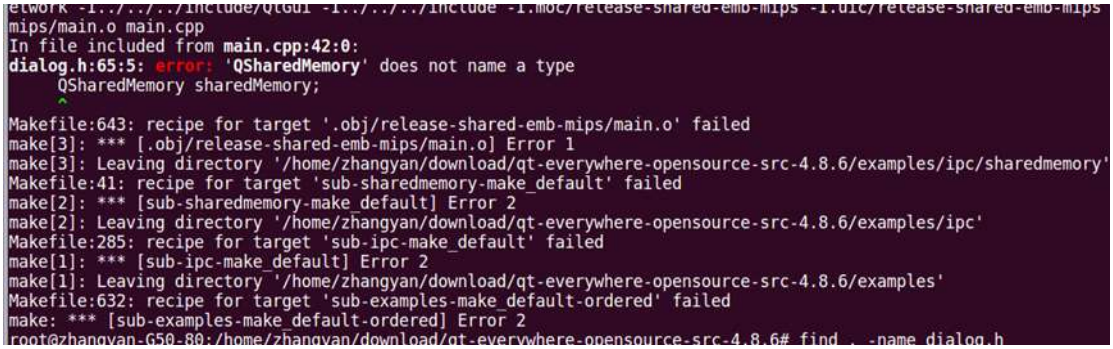
就完成了 Qt 源码的整个编译环节。

但是，很可能编译环节没有这么顺利，会出现报错，这就需要我们按照报错信息去排查错误保证编译的顺利进行。这里举例说明两个比较常见的报错情况：

（1） 提示 mips64el-linux-g++ 命令找不到

我们添加的交叉编译路径是在 HOME 目录下进行的，但是在执行的时候是 sudo make 在 root 目录下进行的，root 目录的.bashrc 没有添加交叉编译工具链的路径，因此会报错。我们可以在 root 目录下按照之前的步骤执行，如果想确认是否添加路径成功，可以在 root 下的 HOME 目录执行：`mips64el-linux-g++ -v` 可以看到版本的打印信息。

（2） 提示“QSharedMemory” does not name a type




我们在编译过程中进行了两次 configure 的声明，因此在调用的时候会出现调用的位置出错的问题，报出类未声明的错误。可以有两种解决办法：

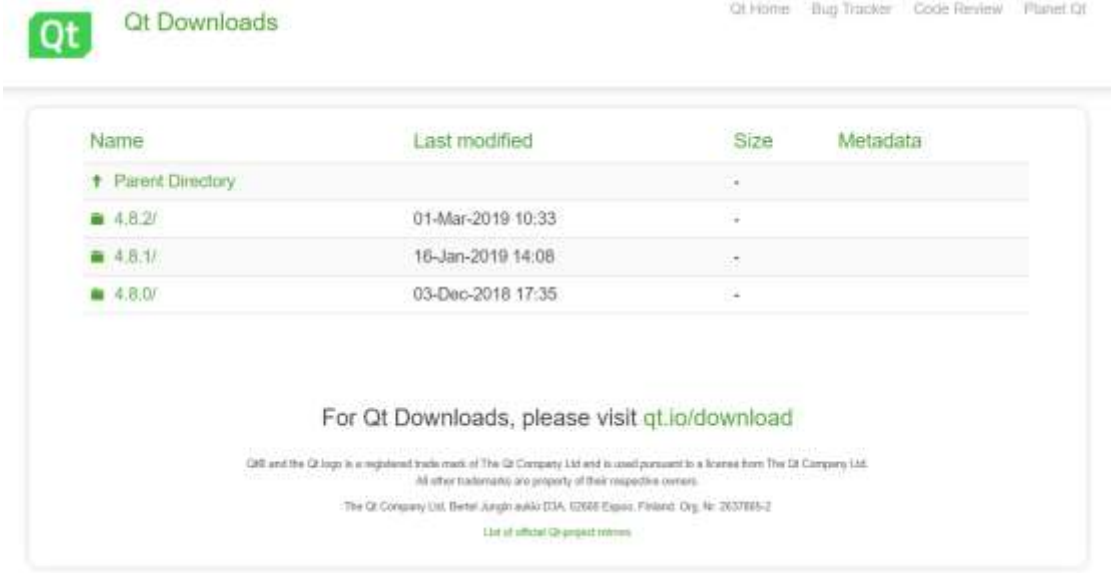
解决方法 1：make confclean 然后再重新./configure 一下。

解决方法 2：在./configure 后面的参数中将-qt-zlib 选项去掉。

### 1.3.7 下载 qt creator 4.8

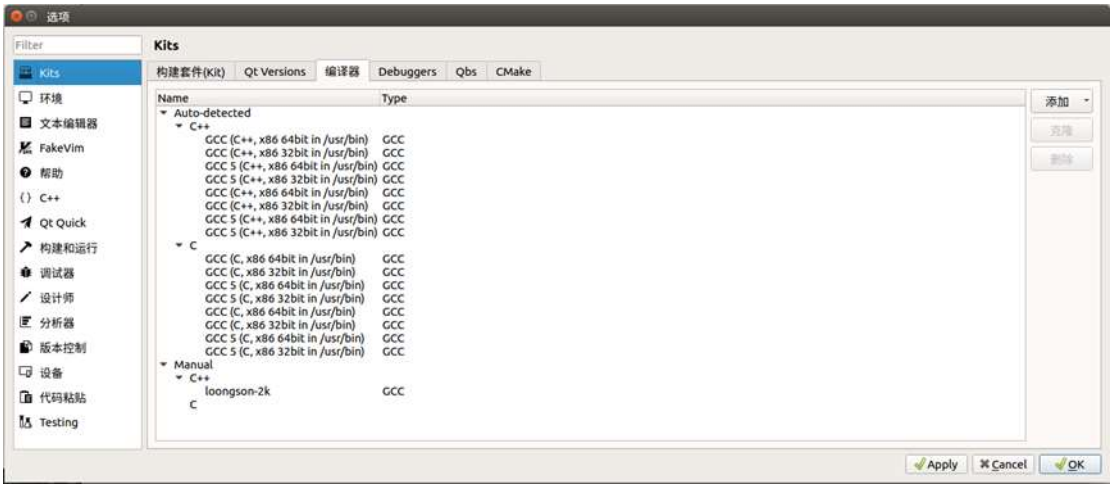
可以直接在终端执行命令 `sudo apt-get install qtcreator` 来进行下载安装，在命令行直接输入 `qtcreator` 打开 qt 界面。

也可以自行在官网上下载安装：https://download.qt.io/official_releases/qtcreator/

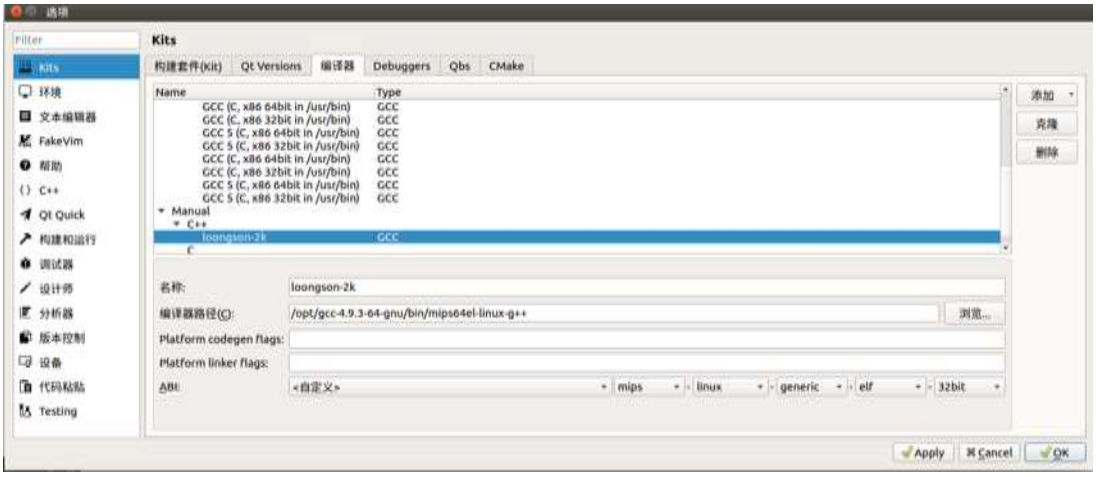


### 1.3.8 在 qtcreator 中配置交叉编译环境

（1） 进入编译器选项，工具->选项->Kits->编译器。

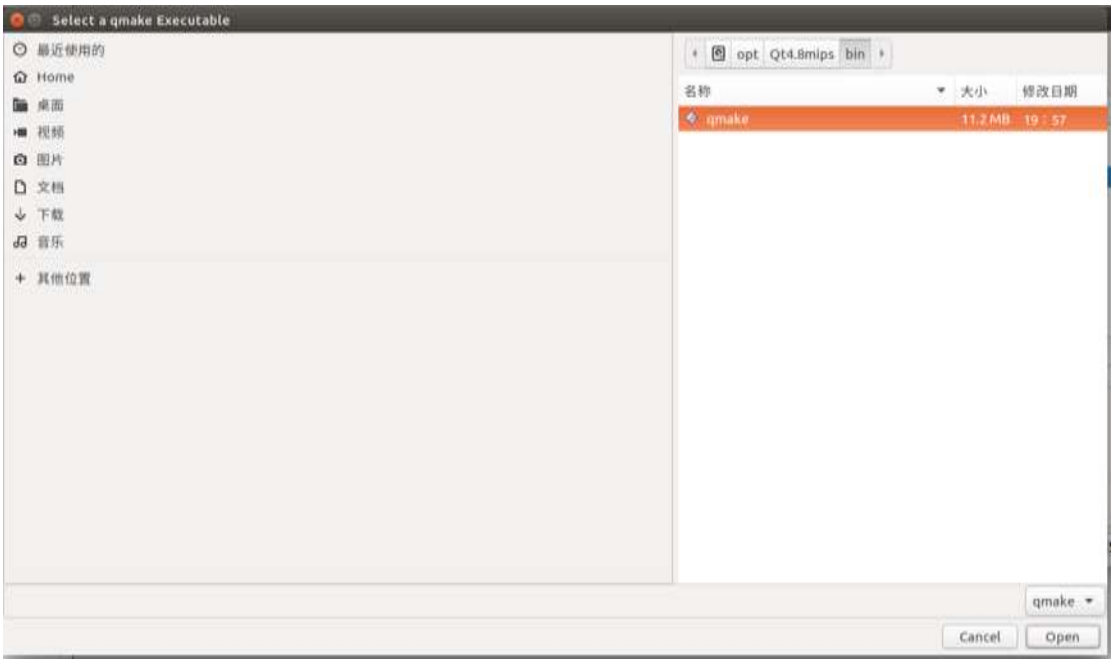



（2） 添加刚刚下载的交叉编译工具链的具体位置。

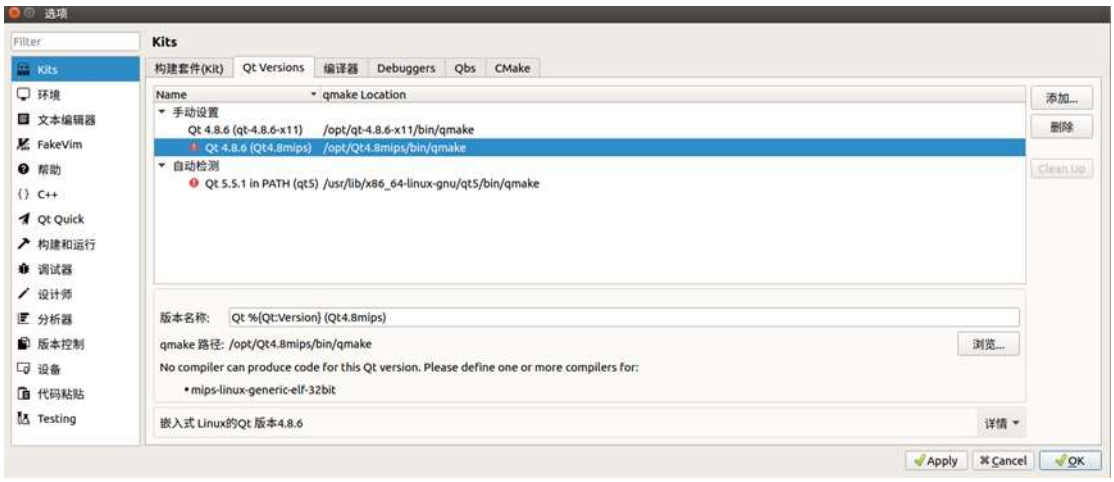



注意：ABI 选项最后要选择 32bit，Qt Versions 是 32bit 的，两者一定要匹配。

（3） 添加 Qt Versions。打开工具->选项->Kits->Qt versions，点击右上角的添加按钮，找到刚我们编译的 qt 源码的安装目录，如果是按照我们的参数， 打开 qmake 路径会得到下图所示的结果：




（4） 可以看到手动设置下面有一个刚添加的 qt Versions，如下图所示:




（5） 设备类型选择通用的 linux 设备，编译器选择自己刚添加的那个，Qt Versions 同样选择自己刚添加的那个，最后点击 Apply 按钮。完成配置。

## 1.4 在 Qt Creator 创建项目

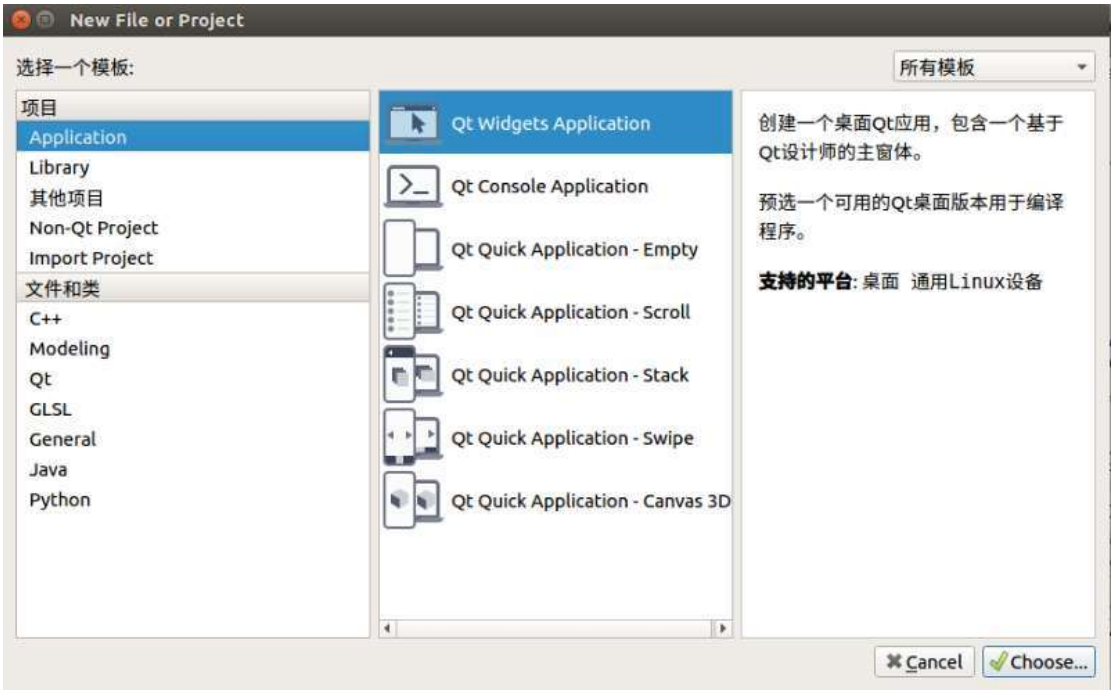
本应用使用 Qt Creator 4.8 进行开发。Qt Creator 是跨平台的 Qt IDE， Qt Creator 是 Qt 被 Nokia 收购后推出的一款新的轻量级集成开发环境(IDE)。此 IDE 能够跨平台运行，支持的系统包括 Linux（32 位及 64 位）、Mac OS X 以及 Windows。Qt Creator 使用强大的 C++ 代码编辑器可快速编写代码，集成了使用浏览工具管理源代码，集成了领先的版本控制软件，包括 Git、Perforce 和 Subversion 开放式文件，无须知晓确切的文件名称或位置搜索类，也不需要文件跨不同位置或文件沿用符号在头文件和源文件，或在声明和定义之间切换。此外，Qt Creator 还集成了特定于 Qt 的功能，如信号与槽 (Signals & Slots)图示调试器， 对 Qt 类结构可一目了然集成了 Qt Designer 可视化布局和格式构建器只需单击一下就可生成和运行 Qt 项目。

在终端中输入命令 `qtcreator` 就可以完成启动，打开界面如图所示：

下面以拼图游戏为例，详解如何在 Qt Creator 中创建项目。

（1） 打开 Qt Creator，选择菜单栏中“文件”→“新建文件或项目”，弹出“新建”对话框。

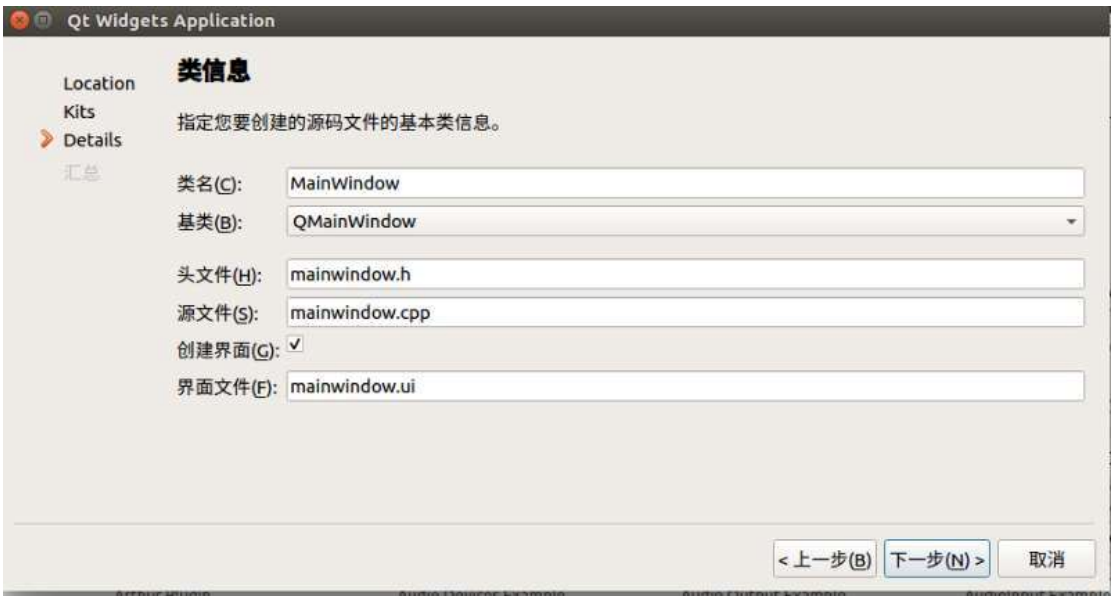
（2） 由于拼图游戏需要有界面、库函数的支持，我们在“Application”中选择“Qt Widgets Application”



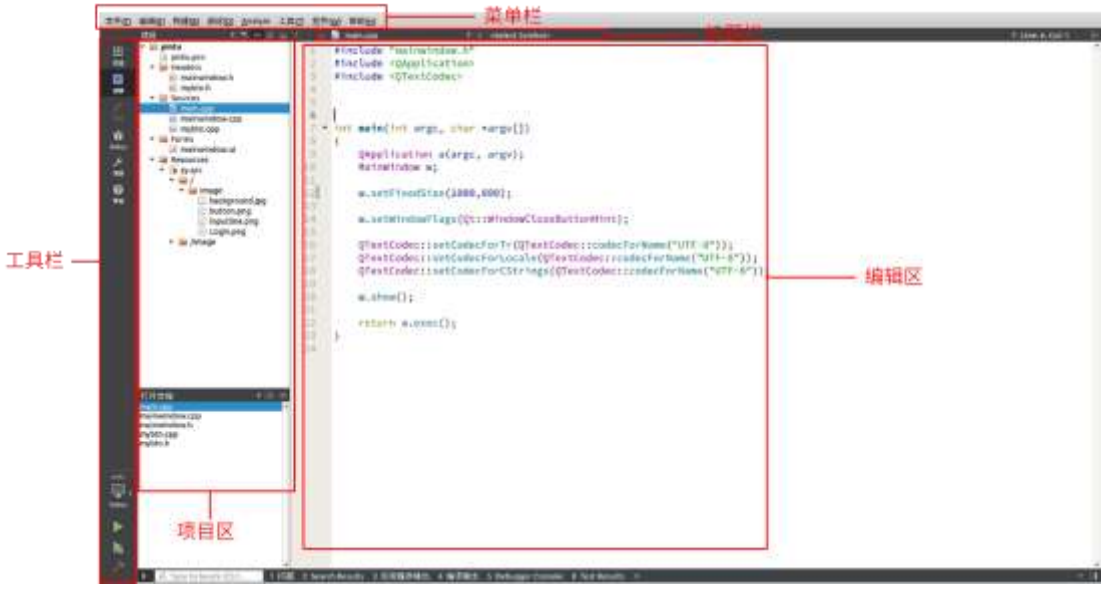

(3) 输入项目名称，选择保存位置。完成后点击“下一步”继续。


(4) 因为应用是在龙芯派上运行的，需要在 Kit 中选择我们之前已经编译好的“Loongson-2k”开发环境。


（5） 在“Detail”界面完成源码文件名，这里直接使用默认文件名就可以了。拼图游戏是有游戏界面的，因此需要勾选“创建界面”选项。在下一步不需要设置直接点击“完成”结束项目新建。




（6） 完成新建后，回到了代码编辑区“mainwindow.cpp”，窗口构成如图所示。




从图中可以看出，Qt Creator 开发界面由菜单栏、标题栏、工具栏、项目区窗口、编辑区组成。简单介绍一下上述界面组件的功能。

（1） 菜单栏：所有的操作选项都可以在菜单栏中找到。

（2） 标题栏：显示项目名称和当前编辑的文件名称。

（3） 工具栏：常用的工具包括编辑、调试、debug、项目和帮助都集成在了侧边和底部的工具栏中。

（4） 项目区窗口：包括整体项目的树序列和文件视图两个部分。

（5） 编辑区：显示当前文件的编辑状态。

## 1.5 主体代码前的准备

在进入主体代码的编写之前，需要做一些提前准备。在头文件 main.cpp 中配置好主体代码 MainWindow 的窗体大小，配置对中文的支持以及按照之前的功能规划预先规划好函数与变量。具体实现代码如下：

UTF-8：UTF 意为“Unicode 转换格式”，后面的数字“8”表明至少使用 8Bit 储存字符。编码规则很简单，如果只有一个字节，那么最高的比特位为 0；如果有多个字节，那么第一个字节从最高位开始，连续有几个比特位的值为 1，就使用几个字节编码，剩下的字节均以 10 开头。

### 1.5.1 头文件配置

主函数 main.cpp 里面需要两件事情，第一是设置 MainWindow 主窗体的大小，横长 1000 个单位长度，纵长 800 个单位长度；第二是在 QTextCodec 中声明以 UTF-8 的格式显示中文。

UTF-8：UTF 意为“Unicode 转换格式”，后面的数字“8”表明至少使用 8Bit 储存字符。编码规则很简单，如果只有一个字节，那么最高的比特位为 0；如果有多个字节，那么第一个字节从最高位开始，连续有几个比特位的值为 1，就使用几个字节编码，剩下的字节均以 10 开

头。

具体实现代码如下：

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;

    w.setFixedSize(1000,800);

    w.setWindowFlags(Qt::WindowCloseButtonHint);

    QTextCodec::setCodecForTr(QTextCodec::codecForName("UTF-8"));
    QTextCodec::setCodecForLocale(QTextCodec::codecForName("UTF-8"));
    QTextCodec::setCodecForCStrings(QTextCodec::codecForName("UTF-8"));

    w.show();

    return a.exec();
}
```

### 1.5.2 函数和主要变量声明

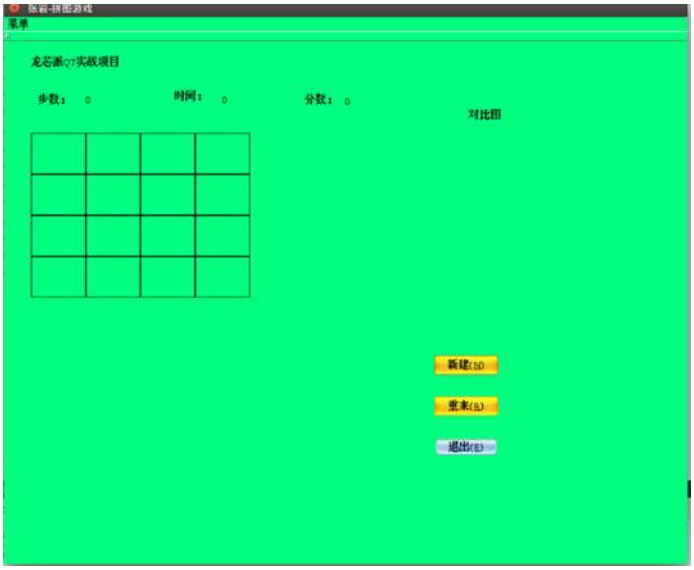
面对对象编程是 C++的重要特性之一，我们需要将最开始定义的业务流程抽象封装为函数，在 mainwindow.cpp 中的头部进行声明。具体代码如下：

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

    void cutImage();//将图片进行分割成一个一个小方块
    void Random();//打乱
    void moveImage();//图片移动
    void mousePressEvent(QMouseEvent *event);//鼠标点击事件
    int taskFinish();//判断是否拼图完成
    void sumPoint();//拼图完成后计算分数
    int sumSteps;//这个变量储存总步数;
    QTimer *timer; //定时器

private slots:      //以下都是一些和信号绑定的槽函数
    void on_btn_clicked();//新建按钮触发的槽函数
```

```
19. void onTimerOut();      //定时器溢出槽函数
20. void on_pushButton_clicked();//退出按钮触发的槽函数
21.     void on_pushButton_2_clicked();//重来按钮触发的槽函数
22.     void on_action_N_triggered();//菜单中新建的槽函数
23.     void on_action_R_triggered();//菜单中重来的槽函数
24.     void on_action_E_triggered();//菜单中退出的槽函数
25.     void on_action_3_triggered();//菜单中初级的槽函数
26.     void on_action_4_triggered();//菜单中中级的槽函数
27. void on_action_5_triggered();//菜单中高级的槽函数
28.
29. private:
30.     Ui::MainWindow *ui;
31.     QString strFileName;//文件名称
32.     QImage* pSourceImage;//原图
33.     QLabel* pLbImage[36];//存储最多 36 个 label，代码中需要将 pImage[x][y]的图片
    设置到相应的 label 上
34.     QImage pImage[6][6]; //存储最多 36 小图片
35.     int pCompare[6][6];  //标记数组；
36. protected:
37.     void keyPressEvent(QKeyEvent *event);
38. };
```

## 1.6 构造游戏界面

在应用的最开始，我们需要通过函数配置，绘制出初始界面，包含点击按钮、计步、计时、计分和图片分割线等部分。如图所示。



### 1.6.1 界面初始化

在 mainwindow.cpp 中的构造函数中对界面进行初始化，类的构造函数是类的一种特殊的成员函数，它会在每次创建类的新对象时执行。具体实现代码如下：

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
```

### 1.6.2 创建定时器，构建计时、计分、计步联系

首先创建一个定时器，作为我们之后开始玩游戏计时使用，然后将定时器的超时信号与槽（功能函数）联系起来。sumSteps 和 tim 以及全局变量 sorce 分别作为记录步数，时间和最终分数的变量，并调用 setText 初始化它们的值都为 0。具体代码如下：

```
ui->setupUi(this);
timer=new QTimer(this);
connect(timer,SIGNAL(timeout()), SLOT(onTimerOut()));
int sumSteps = 0;
int tim = 0;
ui->label_3->setText((QString::number(sumSteps)));
ui->time_label->setText((QString::number(tim)));
ui->sorce_lable->setText((QString::number((sorce))));
```

### 1.6.3 绘制图片分割线

pSourceImage 是我们选择图片的原图 Qimage 指针变量，先把它置为空，设置一个 for 双循环，这个 for 循环根据变量 a 的大小来决定游戏难度，比如是 4x4,5x5,还是 6x6。这里默认 a 的大小为 4。

在循环里面，plbImage 是 Qlable 类型的一个数组，大小为 36，因为游戏难度最大是 6x6，所以最多要有 36 个 lable;每循环过来一次创建一个 lable，并调用 setGeometry 方法设置这个 lable 的起始位置和大小(PHOTO_X 和 PHOTO_Y 是起始位置, SMALL_W 和 SMALL_H 是每个 lable 的大小)，然后再调用 move 方法，将每个 lable 移动到相应的坐标位置，横着排列，最后再调用 setFrameShape 方法为每个 lable 添加边框。到这初始化就完毕了，就出现了上图的田字格。

绘制图片分割线的代码如下：

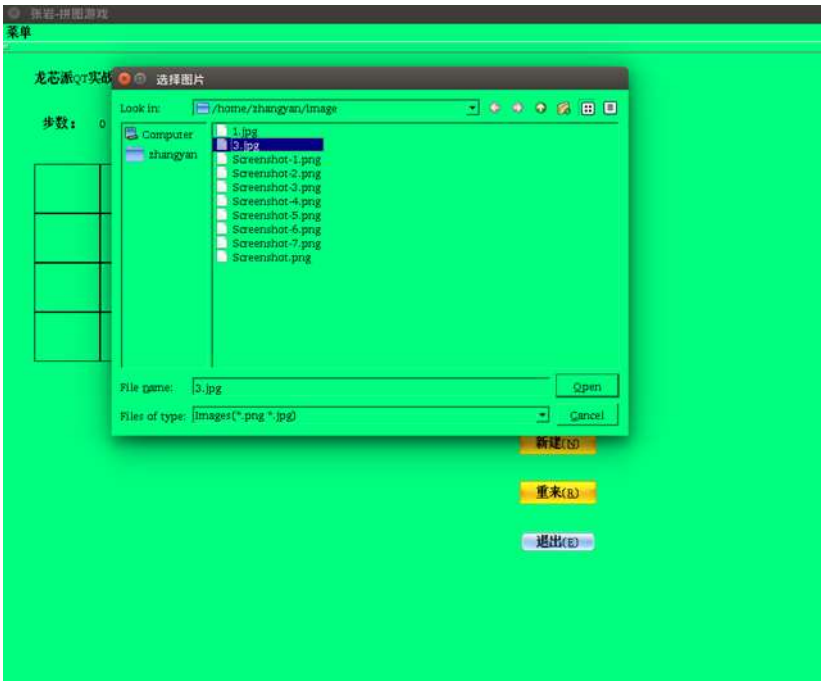
```
pSourceImage=NULL;//指针指向 NULL
for(int i=0;i<a;i++){
        for(int j=0;j<a;j++){
            pLbImage[i*a+j] = new QLabel(this);
            pLbImage[i*a+j]->setGeometry(PHOTO_X,PHOTO_Y,SMALL_W,SMALL_H);
```

```
            pLbImage[i*a+j]->move(PHOTO_X+SMALL_W*i,
                                  PHOTO_Y+SMALL_H*j);
            pLbImage[i*a+j]->setFrameShape(QFrame::Box);
        }
```

### 1.6.4 添加点击按钮

从上图中可以发现，点击按钮“新建”、“重来”和“退出”是链接到动作的。因此，我们需要为这三个按钮添加槽函数，“新建”的槽函数是 on_btn_clicked，“重来”的槽函数是 on_pushButton_2_clicked，“退出”的槽函数是 on_pushButton_clicked。

当用户点击“新建”后就会过来执行相应的槽函数，然后选择一张图片。在槽函数里首先调用 QFileDialog 类的 getOpenFileName 方法，里面有四个参数，第二个参数是打开对话框的标题，第三个参数是打开图片所在的路径 0，第四个参数是图片的类型，如下图所示：




具体实现代码如下：

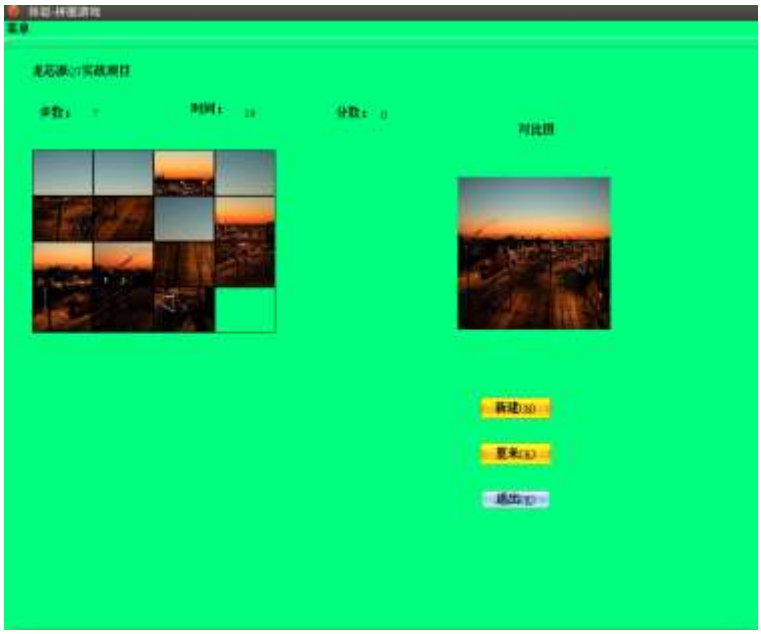
```
void MainWindow::on_btn_clicked()
{
    //打开对话窗
    strFileName =
        QFileDialog::getOpenFileName(this,
                                     "选择图片",
                                     "/home/zhangyan/image",
                                     "Images(*.png *.jpg)");
if(strFileName==NULL) return ;
```

### 1.6.5 放置图片

选择完图片之后，拿到 getOpenFileName 的返回值 strFileName,然后 new 一个图片出来赋值给原图 pSourceImage 指针变量，然后调用 scaled 将原图缩放到为图右方的对比图的尺寸，最后调用 setPixmap 将图片设置上去。具体实现代码如下：

```
1.  if(NULL!=pSourceImage)      //把原来的空间释放掉
2.  {
3.      delete pSourceImage;
4.      pSourceImage = NULL;
5.  }
6.  pSourceImage =new QImage(strFileName);
7.  if(pSourceImage == NULL) return ;
8.
9.  QImage tep=pSourceImage->scaled(ui->label->width(),
10.                                  ui->label->height());
11. gt;label->setPixmap(QPixmap::fromImage(tep));
```

调用 cutImage()函数来将图片分割到田字格中，cutImage()函数的实现将在下一节详解，这样我们就得到了构想中的界面：



## 1.7 拼图动作的实现

### 1.7.1 使用 cutImage()分割图片

在分割之前，我们调用 scaled 方法将图片缩放到整个田字格大小。cutImage()使用了 for

双循环将图片依次按 lable 大小 copy 出来到 pImage 数组里面，然后再将图片设置到相应的 lable 上面，然后 pCompare 数组用来记录每个 lable 对应图片的下标，用于之后移动图片使用；这样就实现了每一个 lable 都对应一个小图片。注意到这里图片还是完整的，并没有被打乱。具体实现代码如下：

```
1.  void MainWindow::cutImage()
2.  {
3.  QImage temp = pSourceImage->scaled(SMALL_W*a, SMALL_H*a);
4.    for(int i=0;i<a;i++){
5.        for(int j=0;j<a;j++){
6.          pImage[i][j]=temp.copy(i*SMALL_W,
                    j*SMALL_H,
7.                                       SMALL_W,
8.                                       SMALL_H);
9.          pLbImage[i*a+j]->setPixmap(QPixmap::fromImage(pImage[i][j]));
10.         pCompare[i][j]=i*a+j;
11.         }
12. }
```

### 1.7.2 挖空图片块

拼图需要在图中挖空一个图块用于其他图块的移动，我们选择将右下角的 lable 图块挖空。具体实现代码如下：

```
1.  voQPixmap tep(":/blank.jpg");
2.  pLbImage[a*a-1]->setPixmap(tep);
```

### 1.7.3 使用 Random()函数打乱图片块

打乱图片以及之后鼠标和键盘移动图块的关键在于找到空白格。因为初始空白格在右下角，自然它对应的下标就是 a*a-1，也就是 15，所以遍历整个田字格，找到下标是 15 的那个 lable 图块，将它的 x,y 坐标找出来。x,y 一开始赋值为-1 是为了如果没有找到，直接退出。

我们将打乱的次数设定为 100 次，以 i 的循环的方式实现打乱的循环。

具体实现代码如下：

```
1.  void MainWindow::Random()//打乱
2.  {
3.      sumSteps=0;
4.      ui->label_3->setText((QString::number(sumSteps)));
5.      tim=0;//设置步数和时间都为 0
6.  ui->time_label->setText((QString::number(tim)));
7.      int x=-1;
```

```
8.      int y=-1;
9.      for(int w=0;w<a;w++){//找到空白格
10.         for(int j=0;j<a;j++){
11.             if(pCompare[w][j]==a*a-1){
12.                 x=w;
13.                 y=j;
14.                 break;
15.             }
16.         }
17.         if(x!=-1)
18.             break;
19.     }
20.     if(x==-1||y==-1)
21.          return ;
22. qsrand(QTime(0,0,0).secsTo(QTime::currentTime()));//使用 qsrand 产生一个随机
    数。
23.     for(int i=0;i<100;i++){
24.         int ret = qrand()%4;//随机将 4 个数字排列，即 0,1,2,3:分别代表将空白格向
    上，向下，向左，向右移动。
25.         switch (ret)
26.         {
27.         case 0://右
28.             if(x<a-1)
29.             {
30.                 pCompare[x][y]=pCompare[x+1][y];
31.                 pCompare[x+1][y]=a*a-1;
32.                 x++;
33.             }
34.             break;
35. //0 代表向右移动，首先如果要是向右移动的话，找到的这个空白格的 x 坐标必须要小于 a-1,
    也就是只能是 0,1,2,因为如果是 3 的话已经在最右边了，不能再向右移动了。然后将空白格和
    它右边的 lable 图片下标进行交换，这时空白格的坐标就变成了 x+1,y,所以要 x++，向左移动
    同理。
36.         case 1://左
37.             if(x>0)
38.             {
39.                 pCompare[x][y]=pCompare[x-1][y];
40.                 pCompare[x-1][y]=a*a-1;
41.                 x--;
42.             }
43.             break;
44.         case 2://下
45.             if(y<a-1)
46.             {
```

```
                pCompare[x][y]=pCompare[x][y+1];
                pCompare[x][y+1]=a*a-1;
                y++;
            }

            break;
//随机到的数是 2 代表向下移动，向下移动的话，空白格的 y 坐标必须要小于 a-1,也就是必须要是 0,1,2，避免越界，然后交换空白格和它下面那个 lable 图片的 pCompile 值，最后记得要 y++，实时更新空白格的坐标位置。向上同理。
        case 3://上
            if(y>0)
            {
                pCompare[x][y]=pCompare[x][y-1];
                pCompare[x][y-1]=a*a-1;
                y--;
            }
            break;
        default:
            break;
        }
}
```

### 1.7.4 使用 moveImage()函数移动图块

Random()函数使用循环打乱只是把每个 lable 图块的下标交换了一下，真正移动图块的是 moveImage()函数，根据打乱后的每个 pCompile 的值，去找到这个值对应的原本的 lable 图块，并重新按顺序一行一行排列下来。

通过一个 for 双循环，依次拿到打乱后的 pCompare 的值，然后在找到这个值对应的 lable 图片，在之前 cutImage 函数中每个 lable 图片都存放 pLbimage 一维数组中，然后再将这个 lable 图片通过调用 move 函数依次按顺序排列。比如 pCompare[0][0]的值通过打乱后由原来的 0 变为 6,那么就将 pLbimage[6]对应的 lable 图片移动到 x=0,y=0 位置处。具体实现代码如下:

```
void MainWindow::moveImage()      //图片移动
{
    for(int i=0;i<a;i++)
    {
        for(int j=0;j<a;j++)
        {
            int index = pCompare[i][j];
            pLbImage[index]->move(PHOTO_X+i*SMALL_W,
                                  PHOTO_Y+j*SMALL_H);
        }
    }
```

```
12. }
```

打乱图片后启动定时器，设置溢出时间为 1000ms，也就是每隔一秒，就执行一次溢出函数 onTimerOut,在溢出函数里记录时间的 tim 变量每次加一，然后设置到 ui 界面的 time_label 上面，这样就实现了每隔一秒的时间累加。具体实现代码如下：

```
1.      if(timer->isActive())
2.      {
3.          timer->stop();
4.      }
5.  timer->start(1000);
6.  void MainWindow::onTimerOut()
7.  {
8.      tim++;
9.      ui->time_label->setText((QString::number(tim)));
10. }
```

## 1.8 为应用添加鼠标事件

完成图块打乱之后，就需要添加鼠标动作事件了。鼠标动作可以分为两部分：①点选要移动的图块；②点击抠出的空图块。这样，需要移动的图块就会向空图块移动。

### 1.8.1 判断拼图状态，关联计步

我们使用 num 作为记录步数的变量，每走一步都要加一。每一次鼠标点击执行之后都要判断是否拼图完成，对应的函数是 taskFinish()，具体实现代码如下：

```
1.  void MainWindow::mousePressEvent(QMouseEvent *event)   //鼠标点击
2.  {
3.      int num=0;
4.      if(pSourceImage==NULL) return ;
5.  if(taskFinish()) return ;
```

### 1.8.2 为鼠标操作添加限定条件

在该应用中，鼠标的动作是有限定规则的，如：需要有点击行为才可以选定图块，在图块选择框内判定为鼠标动作有效等规则。

首先判断限定 QMouseEvent *event 鼠标点击事件的 event 指针的 button()函数必须是 Qt::LeftButton 或者 Qt::RightButton，也就是说只有点击鼠标的左键和右键才会进入这个 if 里面。具体实现代码如下：

```
1.  if(event->button() == Qt::LeftButton
2.          ||event->button() == Qt::RightButton)
```

```
{
    QPoint pressPoint = event->pos();
```

为鼠标点击的位置添加限定，记录鼠标点击位置的 x,y 坐标，使用 if 将 pressPoint.x()和 pressPoint.y()横纵坐标限定在整个图块选择框中。具体实现代码如下：

```
if(pressPoint.x()>PHOTO_X&&pressPoint.y()>PHOTO_Y
    &&pressPoint.x()<PHOTO_X+SMALL_W*a&&pressPoint.y()<PHOTO_Y+SMALL_H*a)
{
    int x = (pressPoint.x()-PHOTO_X)/SMALL_W;    //列数
    int y = (pressPoint.y()-PHOTO_Y)/SMALL_H;     //行数
```

只有用户点击空白格的四个方向的 lable 图片才有用，所以如果要是向左移动的话（这里的向左移动指的是 lable 图片），那么用户肯定是点击的是空白格右边的那个图片，所以要同时满足两个条件，一个是用户点击的 x 坐标必须大于 0,也就是只能是 1,2,3,如果等于 0 已经到最左边的边界了，不能再向左移动了;另一个条件是这个图片的左边是空白格，也就是 pCompare[x-1][y]的值是 15,这个很重要。同时每移动一次都要将记录步数的 num 加 1。向右，向上和向下同理。具体实现代码如下：

```
            if(x>0&&pCompare[x-1][y]==a*a-1)//判断向左移
            {
                pCompare[x-1][y]=pCompare[x][y];
                pCompare[x][y]=a*a-1;
                num++;
            }
            else if(x<a-1&&pCompare[x+1][y]==a*a-1)//判断向右移
            {
                pCompare[x+1][y]=pCompare[x][y];
                pCompare[x][y]=a*a-1;
                num++;

            }else if(y>0&&pCompare[x][y-1]==a*a-1)//判断向上移
            {
                pCompare[x][y-1]=pCompare[x][y];
                pCompare[x][y]=a*a-1;
                num++;
            }else if(y<a-1 && pCompare[x][y+1] == a*a-1)//判断向下移动
            {
                pCompare[x][y+1] = pCompare[x][y];
                pCompare[x][y] = a*a-1;
                num++;
            }
        }
    }
    sumSteps+=num;
```

```
27. ui->label_3->setText((QString::number(sumSteps)));//将移动的步数设置到 ui 界面
    的 label 上。
28. moveImage();//图片移动和打乱时的原理一样，根据下标移动图片，下标没变的不移动。
29.     if(taskFinish()){
30.         timer->stop();
31.         sumPoint();//每走一步调用 taskFinish()来判断是否拼图完成，然后让定时器停
    止，最后调用 sumPoint()函数来计算分数。
32.     }
33. }
```

# 1.9 判断拼图状态

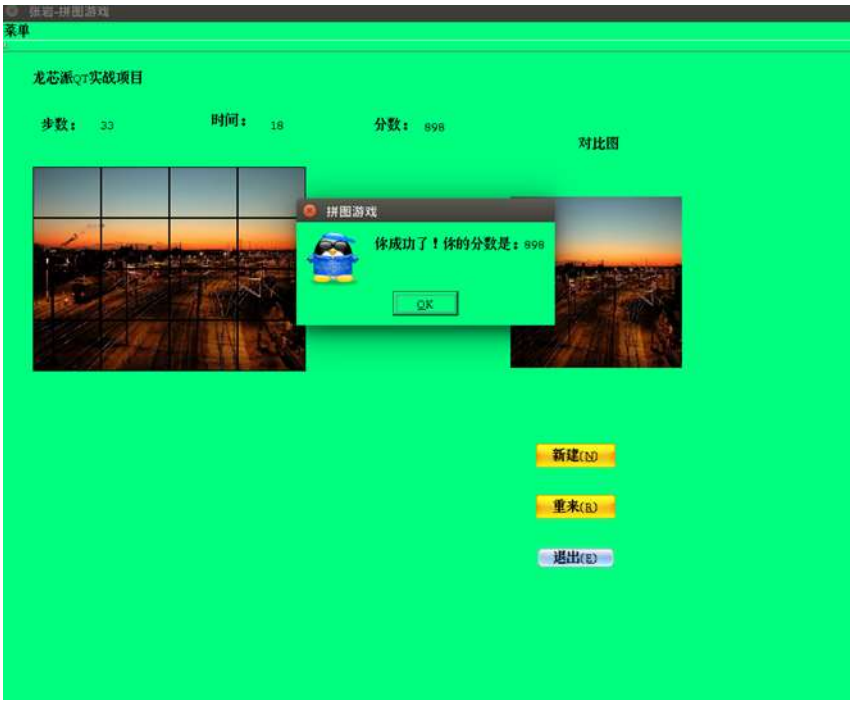
## 1.9.1 判断完成状态

函数 taskFinish()遍历 pCompare 二维数组，看它的值是不是依次按照 0,1,2,3...累加的，这样就和最开始记录的一致。具体实现代码如下：

```
1.  int MainWindow::taskFinish()
2.  {
3.      int y=1;
4.      for(int i=0;i<a;i++)
5.      {
6.          for(int j=0;j<a;j++)
7.          {
8.              if(pCompare[i][j]!=i*a+j)
9.              {
10.                 y=0;
11.                 break;
12.             }
13.         }
14.         if(!y) break;
15.     }
16.     return y;
17. }
```

## 1.9.2 弹出计分对话框

函数 sumPoint ()只会在拼图完成后调用，根据不同的定分规则，来计算分数，然后再弹出一个 QMessageBox 对话框，提示完成分数。如下图所示：




具体实现代码如下：

```
1.  invoid MainWindow::sumPoint ()
2.  {
3.      sorce = 1000 - tim*2 - sumSteps*2;
4.      char buf[64] = {};
5.      sprintf(buf, "你成功了！你的分数是：%d", sorce);
6.      ui->sorce_lable->setText((QString::number((sorce)))); //分数
7.      pLbImage[a*a-1]->setPixmap(QPixmap::fromImage(pImage[a-1][a-1])); //加载
    图片
8.      QMessageBox::about(this, "拼图游戏", buf);
9.  }
```

# 1.10将应用拷贝到龙芯派上

需要明确的是，我们的开发工作是在上位机上完成的，但是程序是在龙芯派使用。因此，这就需要将上位机的代码文件编译为可执行文件，再拷贝到龙芯派执行。

（1） 项目完成之后，在 Qt Creator 内按 CTRL+B 来进行编译，编译完之后在创建项目的路径下，比如我们创建项目的路径如下图所示：


该目录下有一个 build-pintu-Loongson-2K-Debug 目录，如下图所示：


进入到该目录下：


有一个绿色的 pintu 的可执行文件，将该文件拷贝到 U 盘中，然后将 U 盘插入到开发板上。

（2） 将交叉编译后的 qt 源码的安装目录下 lib 目录下的所有库文件拷贝到 U 盘中，就是 make install 时的安装目录，如果是按照之前我们设置的 configure 执行的，那就在下图所示的位置处：


（3） 龙芯派上插上 U 盘后，在龙芯派的 Linux 系统内的命令行终端执行：

```
mount /dev/sdb /mnt
cp /mnt/pintu .
umount /mnt
mkdir /opt/Qt4.8mips/lib   -p
```

然后将我们刚拷贝到 U 盘中的所有的库文件拷贝到 /opt/Qt4.8mips/lib 下。

执行命令

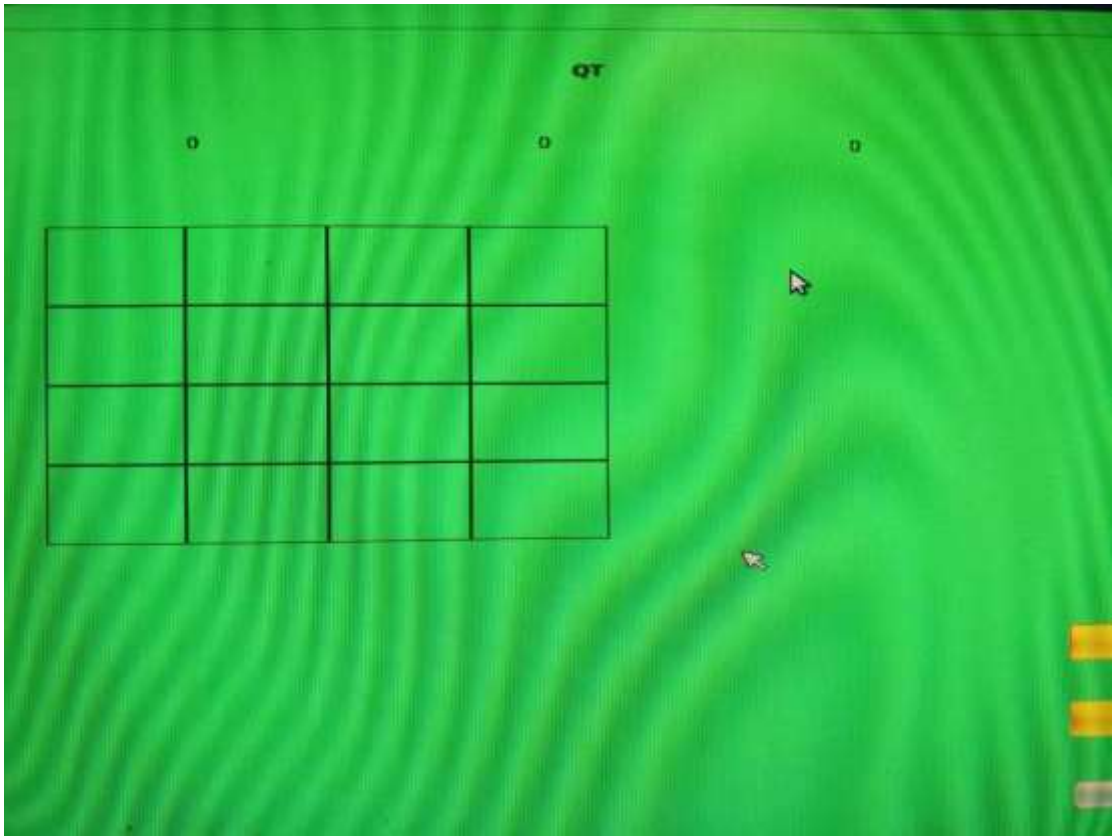
```
./pintu -qws
```

即可以开始游戏。

# 1.11实战演练

## 1.11.1 尝试解决一个 bug

完成了上述代码之后，我们应该可以在龙芯派上开始拼图游戏了，但是很快我们就会发现一个 bug。如下图所示，同一个界面上居然出现了两个鼠标:




这个问题，请想一想如何解决呢?

**原因和解决办法**

出现两个鼠标的原因是，交叉编译的 qt 程序不能在桌面运行。桌面运行下，qt 程序本身会生成一个鼠标，但是操作系统的 GUI 也会有鼠标。这样就出现了两个鼠标同时出现的情况。

解决这个问题的方式非常简单，在龙芯派的操作系统中进入 tty 终端，运行 Qt 程序，就不会出现两个鼠标了。如下图所示:

## 1.11.2 为拼图游戏添加键盘控制

我们的拼图应用已经可以使用鼠标点选的方式控制图块的移动，请试一试在 Qt Creator 内添加键盘事件，使我们可以用设定的键盘按键控制图块的移动完成拼图。

**参考代码**

```
invoid MainWindow::keyPressEvent(QKeyEvent *event)
{
    int num = 0;
    int x=-1;
    int y=-1;
    for(int w=0;w<a;w++){//找到空白格
        for(int j=0;j<a;j++){
            if(pCompare[w][j]==a*a-1){
                x=w;
                y=j;
                break;
            }
        }
        if(x!=-1)
            break;
    }
    if(x==-1||y==-1)
         return;
    switch(event->key())
```

```
    {
        case Qt::Key_Up:
        if(y>0)
        {
            pCompare[x][y] = pCompare[x][y-1];
            pCompare[x][y-1] = a*a-1;
            num++;
            sumSteps+=num;
            ui->label_3->setText((QString::number(sumSteps)));
            moveImage();//图片移动
            if(taskFinish())
            {
                timer->stop();
                sumPoint();
            }
        }
            break;
    case Qt::Key_Down:
        if(y<a-1)
        {
            pCompare[x][y] = pCompare[x][y+1];
            pCompare[x][y+1] = a*a-1;
            num++;
            sumSteps+=num;
            ui->label_3->setText((QString::number(sumSteps)));
            moveImage();//图片移动
            if(taskFinish())
            {
                timer->stop();
                sumPoint();
            }
        }
        break;
    case Qt::Key_Left:
        if(x>0)
        {
            pCompare[x][y] = pCompare[x-1][y];
            pCompare[x-1][y] = a*a-1;
            num++;
            sumSteps+=num;
            ui->label_3->setText((QString::number(sumSteps)));
            moveImage();//图片移动
            if(taskFinish())
            {
```

```
64.                 timer->stop();
65.                 sumPoint();
66.             }
67.         }
68.         break;
69.     case Qt::Key_Right:
70.         if(x<a-1)
71.         {
72.             pCompare[x][y] = pCompare[x+1][y];
73.             pCompare[x+1][y] = a*a-1;
74.             num++;
75.             sumSteps+=num;
76.             ui->label_3->setText((QString::number(sumSteps)));
77.             moveImage();//图片移动
78.             if(taskFinish())
79.             {
80.                 timer->stop();
81.                 sumPoint();
82.             }
83.         }
84.         break;
85.     default:
86.         qDebug() << "wrong!";
87.         break;
88.     }
89. }
```

经过之前代码的分析，相信这个函数看起来就很简单了。注意鼠标和键盘事件函数里都有一个 num 记录步数的变量，这只是个局部变量只记录你鼠标或者键盘每次移动的步数，最后通过 sumSteps += num 来记录到总的步数中。这个函数上来也是先找到空白格，拿到它在 pCompare 里的 x,y 的值，然后一个 switch 语句判断 QKeyEvent *event 键盘事件的 key() 方法，判断具体按的是哪个键，在这里只说一个方向，其它同理。比如空白格向上移动（注意这里操作的是空白格，在鼠标事件里操作的是 lable 图片，因为用户用鼠标点击的是空白格旁边的图片），在 Qt 库中有一个 Qt 类，里面包含了所有键盘的键值，比如 Key_Up，追一下代码你会发现他是一个 16 进制 4 字节的正整数，包含在枚举 Key 里面。然后交换空白格和空白格上面那个 lable 图片的 pCompare 值，记录 num 加一，并记录到总的步数中去，并设置到 ui 界面上对应的步数 lable 上，然后调用 moveImage()来移动图片，移动完后调用 taskFinish()来判断是否完成拼图，相应的是否停止定时器和弹出你的分数。