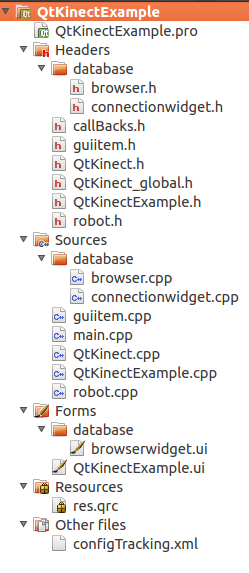
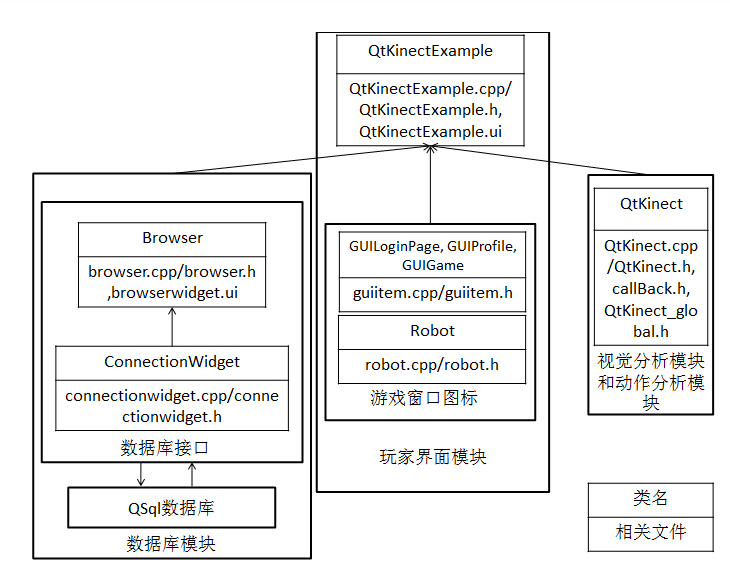
系统的实现基于之前的功能结构设计，由视觉分析模块，动作分析模块，玩家界面模块和数据储存模块组成，以下是系统的整体文件结构。下面的图显示了各个模块的实现文件及其依存关系。





由于该程序由Qt开发，因此文件布局为Qt开发环境标准布局，由.pro编译文件，header头文件文件夹，source文件文件夹，Forms界面文件夹，Resources资源文件夹和其他文件夹组成。下面我首先一一介绍每个模块的实现，然后再介绍基于这些模块，整个游戏过程的实现。

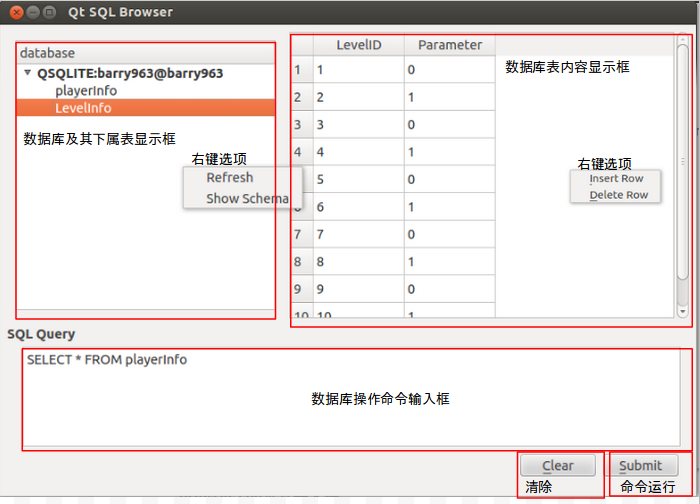
# 数据储存模块

游戏数据库模块用于储存游戏数据，该模块借助Qt的QSql框架实现，运用了许多该框架提供的工具，为程序提供数据库接口，同时为玩家和开发者提供了可视化界面。

数据储存模块由Qt数据库barry963以及database文件夹下源文件browser.cpp，connectionwidget.cpp分别与其头文件，以及界面文件browserwidget.ui组成。browserwidget.ui为数据库界面文件，brower.cpp及其头文件实现了界面文件的功能，并且提供了数据库读写接口，connectionwidget.cpp及其头文件实现了界面中一个窗口的功能。下面将详细介绍每个文件，之后将介绍本游戏数据库的结构设计。

## 界面文件

ui文件描述了数据库窗口的布局，如下图所示。数据库窗口包括了数据库及其下属表显示框，数据库表内容显示框，数据库操作命令输入框和清除以及命令运行按钮。



下面的表格介绍了各个部分的功能

|  |  |
| --- | --- |
| 名称 | 功能 |
| 数据库及其下属表显示框 | 列表显示游戏数据库及其下属表，单击右键会出现刷新和显示数据库元素元类型的选项 |
| 数据库表内容显示框 | 显示左框中选中的数据库表的内容，同时单击右键可以进行插入删除，单击左键可以进行修改 |
| 数据库操作命令输入框 | 输入sql语句命令 |
| 清除按钮 | 清除命令输入框中的字符 |
| 命令运行按钮 | 运行命令输入框中的sql语句命令 |

除了数据库及其下属表显示框是继承了Qt界面基本元素类的ConnectionWidget类的实例，界面上的每一个部分都是Qt界面基本元素类的实例，下面将介绍界面功能的实现。

## connectionwidget.cpp及其头文件

这一组文件实现了界面中数据库及其下属表显示框的功能。在头文件中，定义了继承QWidget的ConnectionWidget类，下图显示了整个类的定义。

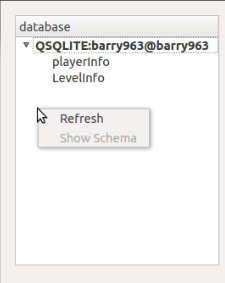


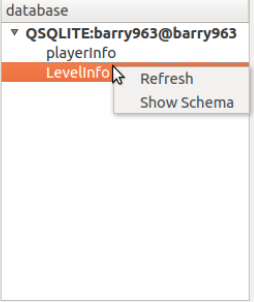
首先看私有成员，其中私有函数void setActive(QTreeWidgetItem \*)函数接受一个指向列表成员的指针，使其所在的数据库字符全黑，表示被选择（如界面文件介绍图所示），同时返回该数据库的标识。另一个私有变量tree为一个指向窗口列表的指针，metaDataAction表示单击右键选择“Show Schema”，即显示元素元类型的动作事件（如界面文件介绍图所示）。activeDb储存了当前激活数据库的标识。

在公有成员中，构造函数初始化了私有成员，将右键选项的点击事件即“Refresh”刷新和“Show Schema”，即显示元素元类型的动作事件，分别与定义的公有槽函数，void refresh() 和void showMetaData()连接起来，从而实现该右键功能。最后，构造函数调用了QMetaObject类下的connectSlotsByName()函数，该函数的作用是，将事件信号与含其名字的槽函数自动连接起来，比方说，一个叫button1的按钮，其单击事件信号clicked()可以自动与button1\_clicked()槽函数连接起来。因此，类中定义的信号void tableActived(const QString &table)和void metaDataRequested(const QString &tableName)将分别会与槽函数XXXX\_tableActivated(const QString &table)和XXXX\_metaDataRequested(const QString &tableName)连接起来。除此之外，槽函数void on\_tree\_itemActivated(QTreeWidgetItem \*item, int column)和void on\_tree\_currentItemChanged(QTreeWidgetItem \*current, QTreeWidgetItem \*previous)将会分别自动与Actived事件信号和Changed事件信号连接起来。

共有成员的另一函数QSqlDatabase currentDatabase() const 则返回activeDb储存的数据库标识所代表的数据库，便于进行后续操作。

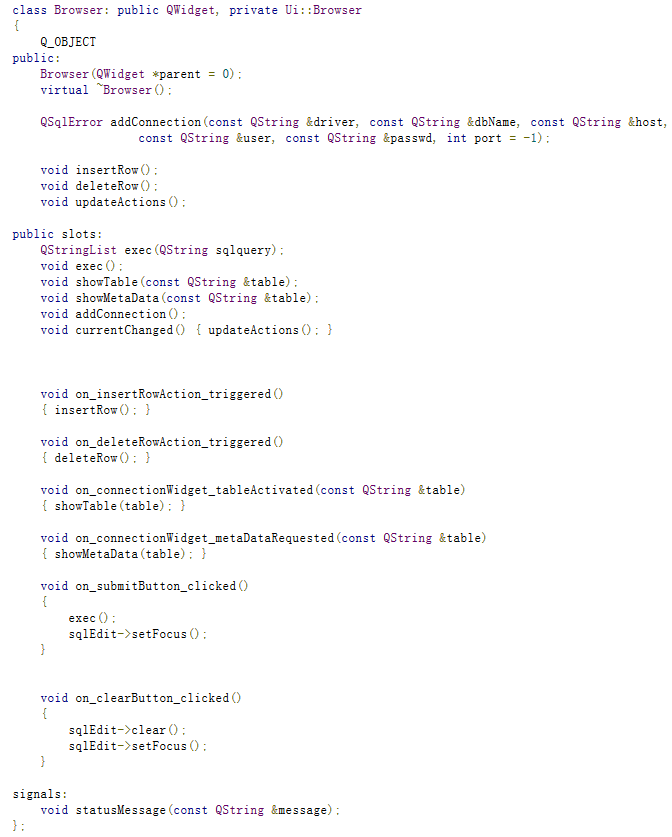
槽函数void on\_tree\_itemActivated(QTreeWidgetItem \*item, int column)的功能是调用setActive()用于item所在列表顶层元素，同时发出tableActived信号。槽函数void on\_tree\_currentItemChanged(QTreeWidgetItem \*current, QTreeWidgetItem \*previous)的功能是判断单击右键时是否允许“Show schema”功能，如果右键选择数据库表则允许，在其他地方右键比如空白处则不允许。





## browser.cpp及其头文件

这一组文件基于上面介绍的ConnectionWidget类，实现了整个数据库窗口的功能。在头文件中定义了Browser类，下图显示了整个类的定义：



公有成员中，构造函数初始化整个界面，还给数据库表内容显示框添加了右键插入选择的功能。QSqlError addConnection(QString&,const QString&,const QString &,const QString&,const QString &,int)函数接受用于连接数据库需要的话参数，返回可能的错误提示信息。void insertRow()，void deleteRow() 和 void updateActions()分别是插入行函数，删除行函数和允许插入删除行的函数。

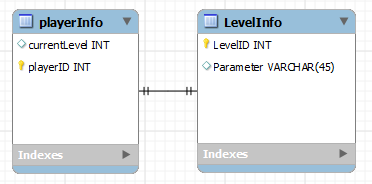
公有槽函数中，QStringList exec(QString sqlquary)函数接受一个储存了sql语句的字符串，执行该语句，同时返回一个QStringList类型的结果。void exec()函数执行数据库操作命令输入框中的sql语句。void showTable(const QString)和void showMetaData(const QString)分别用于显示数据库表内容和显示数据库元素元类型，下面的函数将会直接调用。void addConnection()函数储存了本游戏数据库的登录信息，调用之前的addconnection函数与游戏数据库连接。

void currentChanged()函数直接调用之前的updateActions()函数，与当前行数据改变信号连接。void on\_insertRowAction\_triggered()调用insertRow()函数，与选择插入行动作信号连接。void on\_deleteRowAction\_triggered()调用deleteRow()函数，与选择删除动作信号连接。void on\_connectionWidget\_tableActivated(const QString &table)函数调用showTable()函数，与的ConnectionWidget类中的tableActiveted()信号函数连接，同理void on\_connectionWidget\_metaDataRequested(const QString &table)函数。

信号函数void statusMessage(const QString &message)用于显示所有操作的状态消息，便于指导操作是否成功，需要函数都会调用它。

## 游戏数据库数据表

数据库中有两个数据表，储存玩家信息的playerInfo表和储存关卡信息的LevelInfo，EER图如下所示。playerInfo中主键playerID是玩家信息的唯一标识，同时也是玩家用户名称，currentLevel是玩家当前的关卡。LevelInfo中，主键LevelID是关卡的唯一标识，同时也是关卡名称，Parameter则是关卡参数。连接这个两个表的是LevelID也就是playerInfo中的currentLevel。

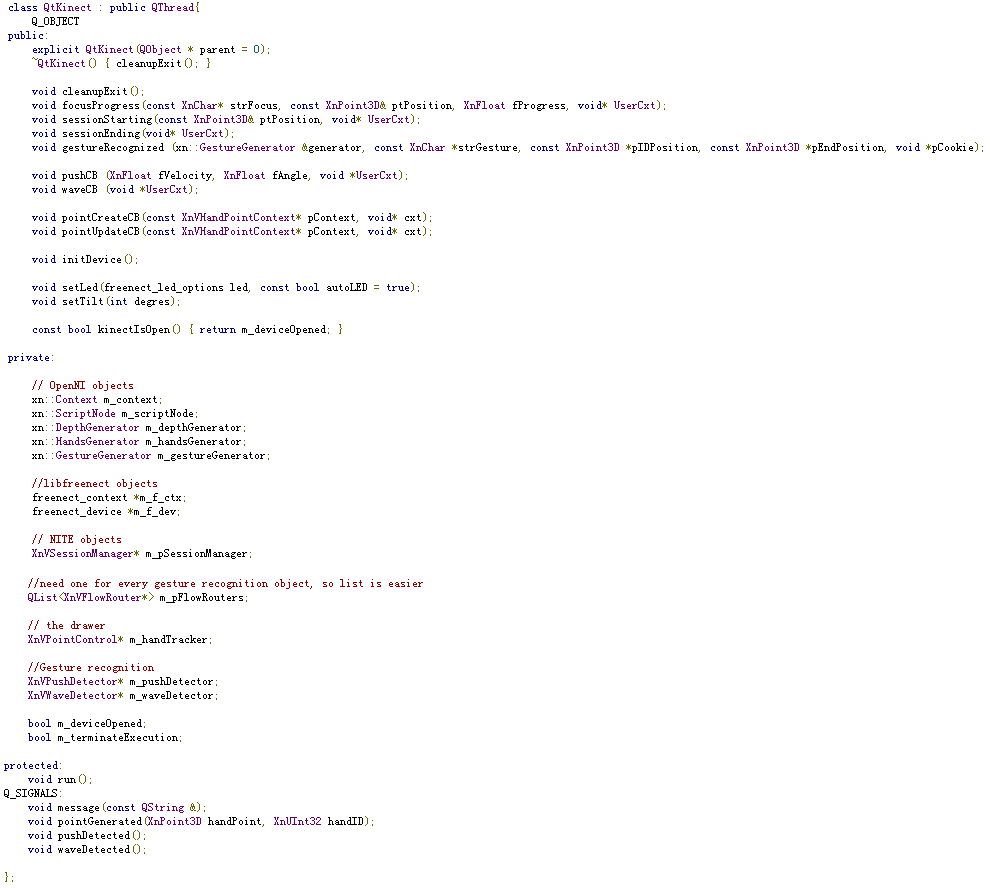


# 视觉分析模块和动作分析模块

这两个模块是整个游戏的核心模块，由源文件QtKinect.cpp及其头文件和callBacks.h实现，实现了包括Kinect硬件初始化，视觉分析，动作分析等功能初始化。

## QtKinect.cpp及其头文件

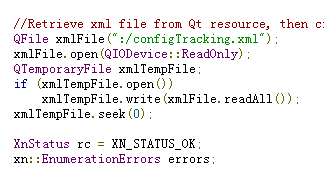
头文件定义了一个QtKinect继承QThread的类，下图显示了整个类的定义



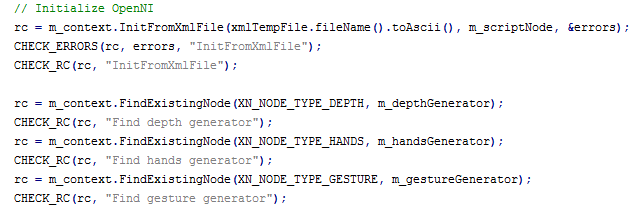
首先看私有成员，m\_context是OpenNI框架中类型为Context的表示场景的类，每一部Kinect设备都需要这样一个类。m\_scriptNode是一个从xml脚本初始化时需要的变量，用来给程序中的对象提供Node节点参考数。m\_depthGenerator是产生深度信息的类，m\_handsGenerator是识别手部信息的类，m\_gestureGenerator是识别动作信息的类。m\_f\_ctx是libfreenect框架下表示场景的类的指针，m\_f\_dev是表示设备硬件信息的类的指针，用来控制LED指示灯和摄像头角度。m\_pSessionManager是NiTE框架中类型为指向XnVSessionManager类的指针，所有的动作识别都需要该变量储存当前的状态。m\_pFlowRouters是类型为XnVFlowRouter的表，每个元素储存一个动作识别对象。m\_handTracker为手部识别对象指针，m\_pushDetector为推手push动作检测对象指针，m\_waveDetector为晃手wave动作检测对象指针，m\_deviceOpened表示Kinect是否打开，m\_terminateExecution表示是否终止命令执行。

信号函数中，message函数用于发出消息信号，消息存于其字符串中，pointGenerated函数用于发出当前手部位置的消息信号，pushDetected和waveDetected函数分别发出检测到push动作和wave动作的消息信号。

公有成员中，构造函数和析构函数分别初始化和释放私有变量的内存。由于本类的作用是完成所有有关Kinect功能的初始化，即完成Kinect的正常启动，供其它模块调用，而initDevice()是其初始化函数，调用了其它成员函数，因此重点介绍initDevice()函数。Kinect的初始化方式分为两种，从xml脚本初始化和直接调用函数初始化，本游戏使用的是xml脚本初始化，该脚本位于other files文件夹中，因此initDevice()函数首先做的就是加载xml文件，如下图所示。xml文件中设置了摄像头视频流深度节点的分辨率以及画面帧数，同时还设置了动作节点和手部节点，原因是本游戏由手部体感配合动作进行操作。



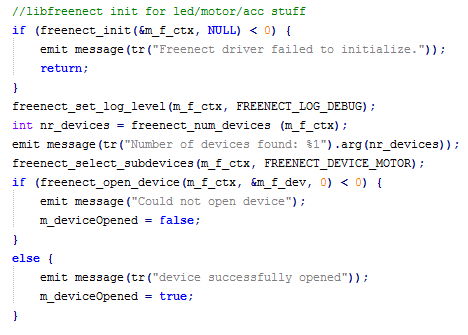
接着，开始初始化OpenNI框架内的对象，如下图所示，初始化了m\_context以及相应的深度信息m\_depthGenerator，手部信息m\_handsGenerator，动作信息的变量m\_gestureGenerator。另外，CHECK\_ERRORS函数和CHECK\_RC函数是QtKinect\_global.h文件中定义的函数，用于检测相应的操作是否存在错误。



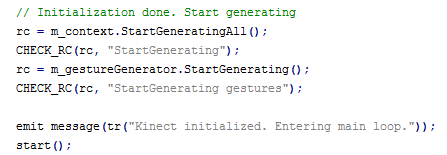
类似的，初始化了NiTE框架下的对象，包括m\_pSessionManager及其管理的push动作检测对象m\_pushDetector和wave动作检测对象m\_waveDetector。之后，将每个动作识别对象添加到m\_pFlowRouters中，全部交给总管理器m\_pSessionManager管理监听。具体来说，对于m\_pSessionManager的初始化，用到了其类函数，RegisterSession，后三个参数分别是会话开始，会话结束和会话聚焦进行的回调函数，用于完整的完成每次会话过程，类似的，后面m\_handTracker，m\_pushDetector和m\_waveDetector都用到了各自的回调函数。回调函数全部都定义在callBacks.h，每个回调函数都调用了本类中各自对应的公有函数，手部检测指针m\_handTracker的回调函数目的是完成手部识别同时不断生成手部坐标，m\_pushDectector和m\_waveDetector的回调函数的目的是完成动作识别，详细可以参加具体定义。



之后，初始化了libfreenect框架中的对象，包括LED指示灯，摄像头角度即马达，同样，每个操作都有相应的报错函数，便于调试，代码如下图。



最后，在初始化了全部变量之后，开始生成动作识别，并且开始运行Kinect。

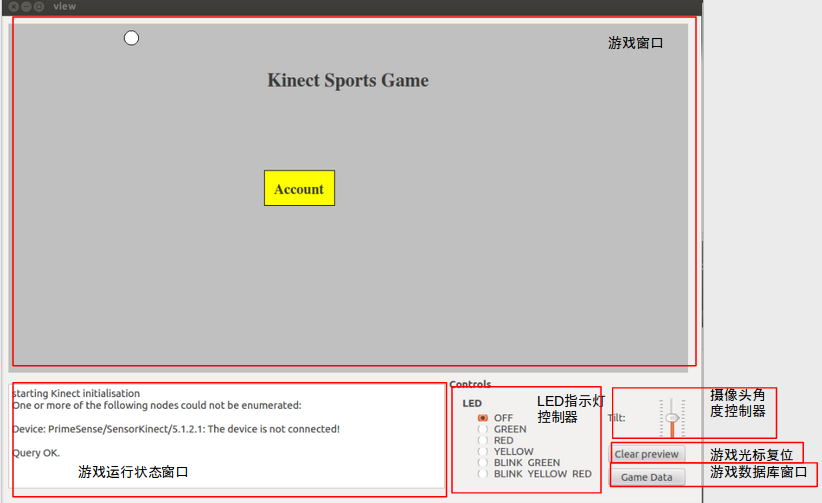


# 玩家界面模块

玩家界面模块是整个游戏的主界面，为玩家提供了与游戏交互的窗口。由Forms文件夹下面的QtKinectExample.ui界面文件以及实现其功能的一个源文件QtKinectExample.cpp及其头文件QtKinectExample.h，还有实现与玩家交互的游戏图标的源文件guiitem和robot及其相应头文件组成。

## 界面文件

ui文件描述了玩家界面的布局，如下图所示。整个游戏界面由游戏窗口，游戏运行状态窗口，LED指示灯控制器，摄像头角度控制器，游戏光标复位按钮以及游戏数据库窗口按钮组成。



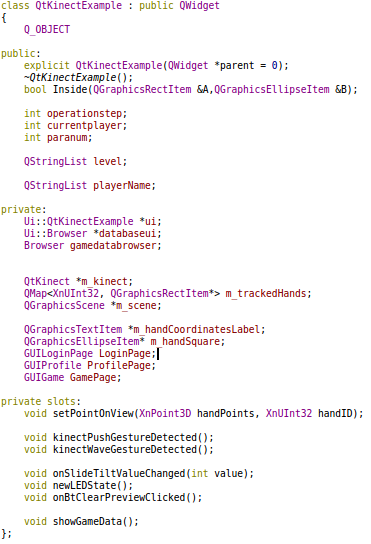
下面的表格介绍了各个部分的功能。

|  |  |
| --- | --- |
| 名称 | 功能 |
| 游戏窗口 | 为玩家提供体感控制操作窗口，包括游戏菜单，体感控制光标等等 |
| 游戏运行状态窗口 | 显示游戏运行的状态信息，比如Kinect初始化信息，人体动作捕捉信息，数据库读取信息，方便游戏调试 |
| LED指示灯控制器 | 用于控制LED指示灯颜色 |
| 摄像头角度控制器 | 用于控制摄像头角度，便于调整到人体动作捕捉的最佳摄像角度 |
| 游戏光标复位按钮 | 用于复位体感操作光标 |
| 游戏数据库窗口显示按钮 | 用于打开游戏数据库窗口，关于游戏数据库下面的章节会有详细介绍 |

界面上的每一个部分都是继承了Qt界面基本元素类的实例，下面将介绍界面功能的实现。

## 源文件QtKinectExample及头文件

这一组文件实现了整个游戏界面的功能，除此之外，还利用上面介绍的模块实现了整个游戏过程。在头文件中，定义了一个公有继承QWidget的QtKinectExample类，QWideget是Qt中所有窗口文件的基类，QtKinectExample整个类的定义如下：

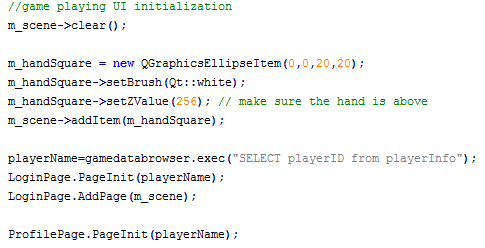


私有数据中，Ui::QtKinectExample类型的ui变量为指向整个界面的指针，含有界面所有元素成员的类，databaseui是指向数据库窗口的指针，gamedatabrowser是数据库模块的类，m\_kinect是完成有关Kinect功能初始化的指向QtKinect类的指针，即前面介绍的视觉分析模块和动作分析模块。m\_trackedHands，m\_hangdCoordinatesLabel以及后面的私有成员都是游戏窗口中游戏体验时的图标文件，用以与玩家交互，在guiitem.h和robot.h中定义，后面将会详细介绍。m\_scene是指向场景类QGraphicsScene的指针，Qt框架中所有的动画都需要该类。

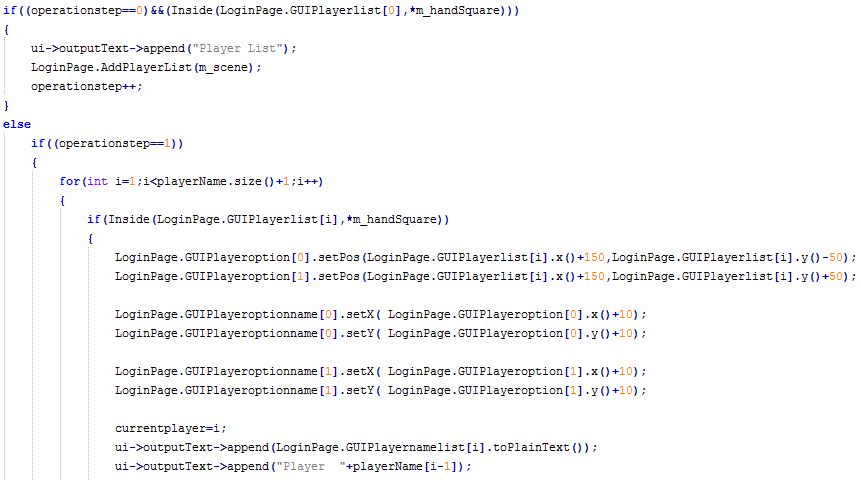
私有槽函数中，void onSlideTiltValueChanged(int value) 函数为摄像头角度控制函数，接受一个有关角度的整型参数。void newLEDState() 函数为LED指示灯控制函数，这两个函数都调用m\_kinect相应的成员函数。void onBtClearPreviewClicked() 函数为游戏光标复位函数，void showGameData() 函数为游戏数据库窗口显示函数。void kinectPushGestureDetected()和 void kinectWaveGestureDetected() 函数分别是检测push动作和wave动作之后的操作函数。

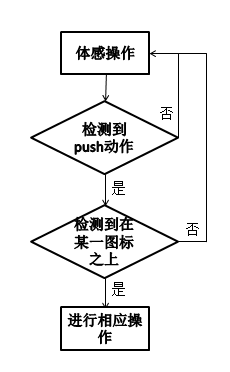
公有成员中，构造函数完成了所有私有成员的初始化，将信号函数和对应的槽函数连接起来，下图显示了全部代码。首先是是连接m\_kinect所指向的QtKinect中定义的信号函数和此类中定义的槽函数连接，包括运行信息，手部位置，动作识别，LED控制，电机控制。之后初始化了m\_kinect和游戏数据库。最后初始化了游戏窗口的图标。

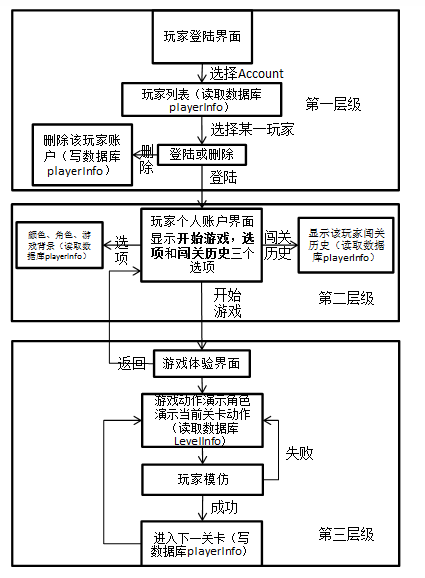




由于游戏的菜单选择都是靠push动作完成，因此，游戏的主要控制流程程序都在检测到push动作之后的槽函数void kinectPushGestureDetected()中，如下图所示。游戏控制的基本原理是，在检测到push动作之后检测当前选择光标的位置是否在对应的图标上（用Inside函数判断），判断是则表明选中此图标，将会进行相应的操作，如下面的流程图所示。代码中的operationstep函数储存了当前界面的层级数，每个层级都有自己的图标，也就会有自己的操作判断，都在代表每个层级的if语句下面，具体的游戏控制流程显示在下面的流程图中。每个层级由guiitem和robot及其相应头文件实现，下面将详细介绍。



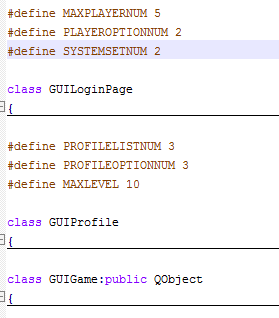




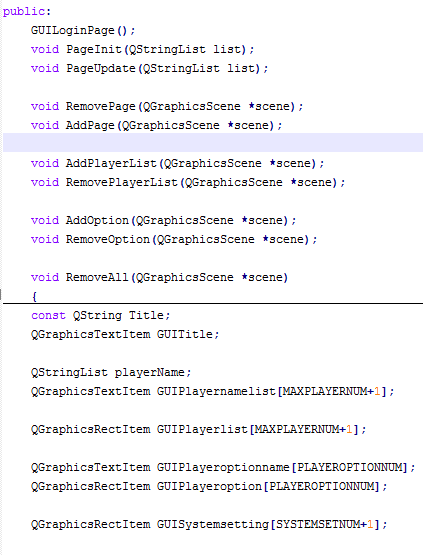
## guiitem和robot及其相应头文件

这个两组文件实现了游戏窗口中的游戏图标，具体来说，guiitem实现了玩家可以控制的游戏菜单，robot实现了游戏体验界面中演示动作的角色。

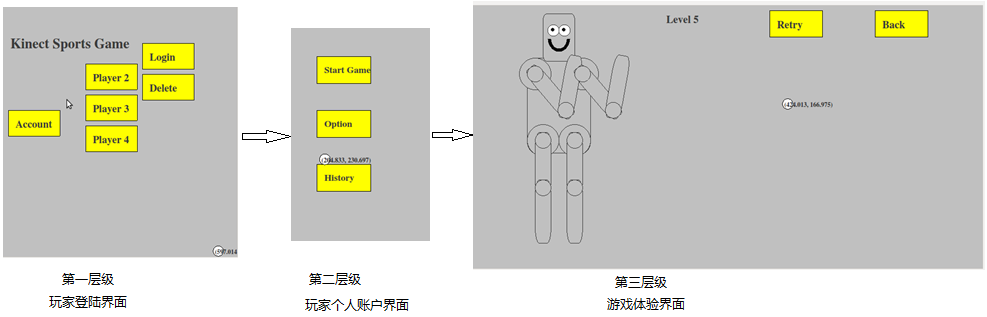
下图展示了guiitem.h中的类定义。该头文件义了三个图标类，分别是游戏登陆界面，玩家个人账户界面和游戏体验界面三个界面层级。



这三个类的设计结构很类似，以游戏登陆界面为例，下图展示了其具体代码。共有成员的对象声明中，有表示每个图标的对象及其上面文字的对象。函数声明中，有整个界面的初始化函数PageInit，它定义了每个图标及其上面文字的位置颜色等属性。PageUpdate函数则用于更新当前图标。下面的函数为相应图标从当前界面添加和移除函数。游戏进行时，每一个界面层级会首先初始化该界面，之后根据玩家操作增加或移除图标。当玩家进入下一个界面层级之后，将会移除当前界面所有图标同时初始化下一个界面层级。



下图是三个界面层级的效果图以及游戏流程图。



下图展示了robot.h中的类定义，它定义了五个类，RobotPart是身体所有部分的基类，RobotHead，RobotTorso和RobotLimb分别表示演示角色身体的各个部分。最后一个类Robot是表示整个演示角色的类。下图展示了每个类所代表的的含义及其相互关系。类中有一个void actionMode(int \*list)函数，它的作用是接受一个代表动作模式的int表的参数，使演示角色做出对应的动作。

