高级语言程序设计大作业实验报告样例

南开大学 计算机大类

姓名 迟文韬

学号 2010387

班级 信安法双学位班

2021年5月9日

高级语言程序设计

实验报告

目录

[高级语言程序设计大作业实验报告](#_Toc21286)

[一. 作业题目](#_Toc29969)

[二. 开发软件](#_Toc18364)

三. 课题要求

[四. 主要流程](#_Toc25240)

[1． 整体流程](#_Toc20452)

[2． 具体实现](#_Toc25354)

[五. 收获](#_Toc23116)

1.功能

2.心得

高级语言程序设计大作业实验报告

**引言：**

为完成本学期的大作业，我在自主学习了一些Qt知识之后，结合相关指导与材料进行可视化编程，制作出一个吃豆人小游戏。主要实现了以下功能：1主页面基本的窗口功能实现2游戏界面静音功能及暂停功能的实现3选关界面在网络已有材料的基础上简单更改，并增添关卡，共可游玩四个关卡4游戏内容上，敌人的智能化追击给游玩者提供了多样化的游戏选择而不仅仅是游戏介绍界面的内容5游戏提供了背景音乐与按键音6实现了按键的动态效果

1. **作业题目**

吃豆人游戏

1. **开发软件**

Qt creator4.3.0

1. **课题要求**

面向对象，图形化编程

1. **主要流程**

***整体流程***

**Main主函数进入mainwindow.cpp运行，完成构造函数和析构函数，搭建主窗口的基本参数并创建ChooseLevelScene与RuleWindow**

**在**RuleWindow**中完成构造函数和析构函数，并用**paintEvent**描绘背景**

**在**ChooseLevelScene**中完成构造函数（进入**gamewindow**）和析构函数，并运行**ChangeButton**与***closeEvent，*

**Gamewindow中创建定时器，用于对象的移动与刷新和效果持续的判断；完成构造函数（调取关卡，布置画面，调取玩家类，敌人类，道具类）和析构函数；构建空格键对应暂停功能，静音键对应静音功能，调用***closeEvent***函数调用**MainGameFunc**进行主游戏过程**

***具体实现***

1头文件

basemoveobject.h物体移动类

baseobject.h物体类

config.h数据配置类

enemyclass.h敌人类（继承自baseobject）

playerclass.h玩家类（继承自baseobject）

propsobject.h道具类（继承自baseobject）

mainwindow.h主界面类

rulewindow.h规则界面类

chooselevelscene.h选关类

gamewindow.h游戏界面类

map.h地图类

mygamescene.h游戏场景类

2源文件（略写构造函数、析构函数、部分函数）

**basemoveobject.cpp**

SetVelocity(int setVelocity)

GetVelocity()

*Move*()

CollideWall()

**baseobject.cpp**

**config.cpp**

在这里可以修改关卡解锁状态

**enemyclass.cpp**

在basemoveobject.cpp 拥有的部分函数基础上增加了trace()

以及形态控制函数:Pause(void)Resume(void)FrightenShift(void)

**playerclass.cpp**

*keyPressEvent*(QKeyEvent \*event)*keyReleaseEvent*(QKeyEvent\*event)对wasd进行事件重写

CleanKeyPress(void)

形态控制函数:Pause(void)Resume(void)Die(void)

拥有basemoveobject.cpp 的部分函数

**propsobject.cpp**

SuperBean(int x, int y)

SlowDownPacman(int x, int y)

定义了位置等形式参量，效果则在gamewindow中描述

**mainwindow.cpp**

**rulewindow.cpp**

*paintEvent*(QPaintEvent \*event)直接将ps制作的一张图片作为界面

**chooselevelscene.cpp**

ChangeButton(void)改变关卡图标的状态，实现选择后续关卡

**的功能gamewindow.cpp**

*keyPressEvent*(QKeyEvent\*event)重写了空格键的暂停功能（通过PauseShift(void)实现）

*closeEvent*(QCloseEvent \*event)重写关闭窗口事件

MainGameFunc()在构造函数中与游戏主进程timer作为游戏核心

AddPropsObject()增添道具的函数

**map.h**

提供地图的原型，在这里可以新增关卡，不过需要相应地在gamewindow、map、mygamescene中添加参数

**mygamescene.h**

1. **收获**

**配合**map.cpp

#define N 3

#define A 2

#define B 1

#define O 0

#define S -1

const int Stage\_1::initData[MAPLENGTH][MAPLENGTH] = {

......

};**实现**

**全地图进行背景绘制**mygamescene.cpp

painter->setBrush(QBrush(Qt::black));

painter->drawRect(-BRICK\_SIZE\*MAPLENGTH/2,-BRICK\_SIZE\*MAPLENGTH/2,BRICK\_SIZE\*MAPLENGTH\*2,BRICK\_SIZE\*MAPLENGTH\*2);

QPixmap GreyStone=QPixmap(":/Images/Brick\_5.png").scaled(QSize(BRICK\_SIZE,BRICK\_SIZE));

for(int i=0;i<MAPLENGTH;++i)

for(int j=0;j<MAPLENGTH;++j)

if (map->WallData[i][j]) painter->drawPixmap(j\*BRICK\_SIZE,i\*BRICK\_SIZE,GreyStone);

break;

**优化键盘操作**playerclass.cpp

**重写了键盘事件**

void PlayerClass::*keyPressEvent*(QKeyEvent \*event)

{

//如果按下指令是由第一次按下产生的（而非长按导致的重复按压与释放指令）

if (!event->isAutoRepeat())

{

//将方向指令输入缓存移动指令中，按压键盘数加一，并记录按压的键盘

switch(event->key())

{

case Qt::Key\_A:

{

Buffer\_Dir\_X=DIR\_VALUE[DIR\_LEFT][0]; Buffer\_Dir\_Y=DIR\_VALUE[DIR\_LEFT][1]; Buffer\_Dir=DIR\_LEFT; ButtenPressed[DIR\_LEFT]=true; ++PressedKeys;

break;

}

case Qt::Key\_D:

{

Buffer\_Dir\_X=DIR\_VALUE[DIR\_RIGHT][0]; Buffer\_Dir\_Y=DIR\_VALUE[DIR\_RIGHT][1]; Buffer\_Dir=DIR\_RIGHT; ButtenPressed[DIR\_RIGHT]=true; ++PressedKeys;

break;

}

case Qt::Key\_W:

{

Buffer\_Dir\_X=DIR\_VALUE[DIR\_UP][0]; Buffer\_Dir\_Y=DIR\_VALUE[DIR\_UP][1]; Buffer\_Dir=DIR\_UP; ButtenPressed[DIR\_UP]=true; ++PressedKeys;

break;

}

case Qt::Key\_S:

{

Buffer\_Dir\_X=DIR\_VALUE[DIR\_DOWN][0]; Buffer\_Dir\_Y=DIR\_VALUE[DIR\_DOWN][1]; Buffer\_Dir=DIR\_DOWN; ButtenPressed[DIR\_DOWN]=true; ++PressedKeys;

break;

}

default:

QGraphicsPixmapItem::*keyPressEvent*(event);

break;

};

}

//调用基类的键盘按压事件函数

}

void PlayerClass::*keyReleaseEvent*(QKeyEvent\*event)

{

//如果释放指令是由第一次释放产生的（而非长按导致的重复按压与释放指令）

//减少键盘按压个数并将对应键盘按压标记清零

if(!event->isAutoRepeat())

switch(event->key())

{

case Qt::Key\_A:

ButtenPressed[DIR\_LEFT]=false; --PressedKeys;

break;

case Qt::Key\_D:

ButtenPressed[DIR\_RIGHT]=false; --PressedKeys;

break;

case Qt::Key\_W:

ButtenPressed[DIR\_UP]=false; --PressedKeys;

break;

case Qt::Key\_S:

ButtenPressed[DIR\_DOWN]=false; --PressedKeys;

break;

default:

//调用基类的键盘释放事件函数

QGraphicsPixmapItem::*keyReleaseEvent*(event);

}

if (PressedKeys<0) PressedKeys=0;

//如果按压键盘数为零则缓存移动指令清零

if(!PressedKeys) Buffer\_Dir\_X=Buffer\_Dir\_Y=0;

//否则说明当前还有键盘在按压

else

{

//将搜索到的第一个仍在按压的键盘对应的指令输入缓存移动指令中

for(int i=0;i<4;++i)

if (ButtenPressed[i])

{

Buffer\_Dir\_X=DIR\_VALUE[i][0];

Buffer\_Dir\_Y=DIR\_VALUE[i][1];

Buffer\_Dir=i;

break;

}

}

}

**绘制原点偏移半个图像大小**

**使得游戏体验更加顺滑，避免交界处角色“卡墙”的问题。**

**例子一**playerclass.cpp

SlowDownPacman::SlowDownPacman(int x, int y) : PropsObject(\_SLOWVIRUS)

{

//将道具类的绘制原点进行偏移，偏移量为半个图像的大小，这样道具类的实际位置刚好处于图像中央

setOffset(-PROP\_SIZE/2,-PROP\_SIZE/2);

setPixmap(QPixmap(":/Images/Virus.png").scaled(PROP\_SIZE, PROP\_SIZE));

X = x; Y = y;

setPos(Y\*BRICK\_SIZE+HALF\_BRICK\_SIZE,X\*BRICK\_SIZE+HALF\_BRICK\_SIZE);

}

setFlags(QGraphicsItem::ItemIsFocusable);//将玩家类设为可聚焦(focus)，这样才能接收键盘信号

setOffset(-PLAYER\_SIZE/2,-PLAYER\_SIZE/2);//将玩家类的绘制原点进行偏移，偏移量为半个图像的大小，这样玩家类的实际位置刚好处于图像中央

**例子二**propsobject.cpp

setPos(P\_COL\*BRICK\_SIZE+HALF\_BRICK\_SIZE,P\_ROW\*BRICK\_SIZE+HALF\_BRICK\_SIZE);

SetVelocity(INIT\_PLAYER\_VELOCITY);

X=P\_ROW; Y=P\_COL;

**对应方向移动的实现**

config.h

const int DIR\_VALUE[4][2]={{0,-1},{-1,0},{0,1},{1,0}};

enemyclass.cpp/playerclass.cpp/

extern const int DIR\_VALUE[4][2];

**敌人的移动函数**

void EnemyClass::*Move*()

{

......

//判断当前方向的垂直方向是否可以移动，如果可以则表示到达了分叉口，随机步长清零

int tempDir\_X=Dir\_X,tempDir\_Y=Dir\_Y;

Dir\_X=DIR\_VALUE[(Dir+1)%4][0],Dir\_Y=DIR\_VALUE[(Dir+1)%4][1];

if (!CollideWall()) LastSteps=0;

Dir\_X=DIR\_VALUE[(Dir+3)%4][0],Dir\_Y=DIR\_VALUE[(Dir+3)%4][1];

if (!CollideWall()) LastSteps=0;

Dir\_X=tempDir\_X; Dir\_Y=tempDir\_Y;

//如果随机步长为零，表明需要转向了

if(LastSteps<=0)

do

{

//随机选择一个方向直到该方向可以移动，并通过数学方法将后退的概率降低

chg=((qrand()%16+4)/5+2)%4;

Dir\_X=DIR\_VALUE[(Dir+chg)%4][0];

Dir\_Y=DIR\_VALUE[(Dir+chg)%4][1];

}while(CollideWall());

//将当前方向更改为新随机方向

Dir=(Dir+chg)%4;

if(Map->RoadData[X][Y]==-1)

{

Dir=DIR\_UP;

Dir\_X=DIR\_VALUE[Dir][0];

Dir\_Y=DIR\_VALUE[Dir][1];

if(CollideWall()) Dir\_X=Dir\_Y=0;

}

}

}

}

**玩家的移动函数**

void PlayerClass::*Move*()

{

//如果缓存移动指令与当前移动指令在同一条直线上，或者已经到达了方块中心

if ( (Buffer\_Dir\_X==Dir\_X || Buffer\_Dir\_Y==Dir\_Y ) || Ctrl)

{

//将缓存指令写入当前移动指令中

Dir\_X=Buffer\_Dir\_X;

Dir\_Y=Buffer\_Dir\_Y;

//如果缓存移动指令的方向与当前方向不同，说明产生了变向

if (Dir!=Buffer\_Dir)

{

setRotation((Buffer\_Dir+2)%4\*90);//改变动图 Dir=Buffer\_Dir;

}

}

BaseMoveObject::*Move*();

}

void PlayerClass::*keyPressEvent*(QKeyEvent \*event)

{

//如果按下指令是由第一次按下产生的（而非长按导致的重复按压与释放指令）

if (!event->isAutoRepeat())

{

switch(event->key())

{

case Qt::Key\_A:

{

Buffer\_Dir\_X=DIR\_VALUE[DIR\_LEFT][0]; Buffer\_Dir\_Y=DIR\_VALUE[DIR\_LEFT][1]; Buffer\_Dir=DIR\_LEFT; ButtenPressed[DIR\_LEFT]=true; ++PressedKeys;

break;

}//键盘方向键的定义

......

default:

QGraphicsPixmapItem::*keyPressEvent*(event);

break;

};

}

}

**自控的“智能”敌人**

20世纪足足有超过10亿人玩过的《吃豆人》被称作现代游戏AI鼻祖。

1980 年 5 月22日，南梦宫发布了《吃豆人》系列的首款游戏，它第一次登陆是在街机上，由于可爱的圆脸形象和简单的操作备受欢迎。四十年后的今天，吃豆人已经成为电子游戏中符号般的存在，出现在各种衍生作品和影视剧中。这款在今天看来有些简单的游戏其实有很多超前的设计，比如不同颜色的鬼魂分别遵循不同的运动规律，是最早的游戏 AI 之一。关于吃豆人游戏的高分竞争，还间接影响了电子竞技的诞生。

基本的移动模式：一方面处理与player的位置关系，一方面处理与墙体的关系，通过随机数实现转向的随机性与前进步数的的随机性。

void EnemyClass::Move(){}

追踪敌人的函数实现

bool EnemyClass::trace(){}

**心得：**

在这次制作过程中，我了解了Qt可视化编程的基本用法，对编程语言的实际应用增添了极大的兴趣。结合所制作游戏的历史渊源我深刻体会到了计算机技术四十年来迅猛的发展。

在实际制作过程中我也遇到了许多问题，由于对Qt庞大功能体系的不熟悉，许多功能的实现并没有理解的透彻，代码编写过程中不少疑难点是在一定的指导下完成的，然而正是这样的编写经历让我对Qt和c++面向对象的编程的理解深入了许多，未来在学习过程中希望能锻炼好自己的基本功，有所提升。