**面向对象方法程序设计与实践报告**

**题 目 黄金矿工**

**序 号 44**

**学 号 20242081402**

**姓 名 李军键**

**任课教师 姜厚云**

**成 绩**

大连理工大学软件学院

2025年7月

**目录**

[评分细则及标准 2](#_Toc143947979)

[1 技术调研报告 3](#_Toc143947980)

[1.1 第一周学习总结（除去代码，不少于1000字） 3](#_Toc143947981)

[1.1.1 内容简介 3](#_Toc143947982)

[1.1.2 难点和解决办法 3](#_Toc143947983)

[1.1.3 学习案例 3](#_Toc143947984)

[1.2 第二周学习总结（同上） 3](#_Toc143947985)

[1.2.1 内容简介 3](#_Toc143947986)

[1.2.2 难点和解决办法 3](#_Toc143947987)

[1.2.3 学习案例 4](#_Toc143947988)

[2 项目开发报告 5](#_Toc143947989)

[2.1 项目简介 5](#_Toc143947990)

[2.2 需求分析 5](#_Toc143947991)

[2.2.1 功能模块 5](#_Toc143947992)

[2.2.2 需求说明 6](#_Toc143947993)

[2.3 系统设计 6](#_Toc143947994)

[2.3.1 界面设计（UI） 6](#_Toc143947995)

[2.3.2 流程设计 6](#_Toc143947996)

[2.3.3 类设计（可选） 8](#_Toc143947997)

[2.4 系统实现 8](#_Toc143947998)

[2.4.1 实现效果 8](#_Toc143947999)

[2.4.2 核心代码 9](#_Toc143948000)

[2.5 系统测试 11](#_Toc143948001)

[2.6 项目总结 11](#_Toc143948002)

**评分细则及标准**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **考察项目** | **总分** | **评分细则（总分40分）** | **分数** | **得分** |
| **技术调研（第一周）** | 20分 | 调研报告 | 10分 |  |
| 平时表现 | 10分 |
| **技术调研（第二周）** | 20分 | 调研报告 | 10分 |  |
| 平时表现 | 10分 |
| **项目开发评分（总分60分）** | | | |  |
| 问题规模 | 10分 | 创新超额完成指定任务，工作量饱满 | 8-10分 |  |
| 基本完成指定任务，工作量一般 | 6-8分 |
| 指定任务未完成，工作量不足 | 0-5分 |
| 技术难度 | 10分 | 模型设计合理优化，采用合适的方法实现 | 8-10分 |  |
| 模型设计基本正确，采用较合适的方法实现 | 6-8分 |
| 模型设计存在问题，采用方法不合理 | 0-5分 |
| 实现程度 | 20分 | 系统实现完整，界面友好，测试全面无误 | 8-10分 |  |
| 系统实现完整，界面友好，存在少许错误 | 6-8分 |
| 系统实现不完整，界面不友好，存在错误 | 0-5分 |
| 报告质量 | 10分 | 报告完整、格式统一、结构清晰、图表正确 | 8-10分 |  |
| 报告较为规范、结构较为清晰、图表基本正确 | 6-8分 |
| 内容不完整不规范、结构不清晰、图表有错误 | 0-5分 |
| 项目汇报 | 10分 | 阐述清晰准确，回答问题准确到位 | 8-10分 |  |
| 阐述基本准确，回答问题基本准确 | 6-8分 |
| 阐述不够清晰完整，回答问题不准确 | 0-5分 |
| 最终得分 | 100分 |  | | |

相关材料（电子报告及项目源码压缩包）的百度网盘链接：

通过网盘分享的文件：44-20242081402-李军键.rar

链接: https://pan.baidu.com/s/1eVc4rclPmVxmSiR90kFDVw?pwd=yhh9 提取码: yhh9

# 1 技术调研报告

## 1.1 第一周学习总结（不少于1000汉字）

### 1.1.1 内容简介

本周主要学习了 EasyX 绘图库的基础知识，核心围绕 “搭建环境 - 理解基础概念 - 掌握绘图功能” 展开，包括：

（1）开发环境搭建：学习了如何安装 Visual Studio（推荐 2022 社区版）及 EasyX 绘图库。

基本概念：

绘图窗口与设备：通过initgraph函数创建绘图窗口，支持设置窗口大小和特殊模式（如带控制台的EX\_SHOWCONSOLE）；设备分为默认绘图窗口和IMAGE对象，可通过SetWorkingImage切换当前绘图目标。

坐标系统：包括物理坐标（原点在左上角，X 轴右正、Y 轴下正，不可修改）和逻辑坐标（默认与物理坐标一致，可通过setorigin设置原点、setaspectratio调整缩放比例或坐标轴方向）。

颜色表示：支持 4 种方式（预定义常量如BLUE、16 进制0xBBGGRR、RGB宏、HSLtoRGB转换），通过setlinecolor等函数设置绘图颜色。

核心绘图功能：

图形绘制：掌握了 33 种基础图形函数，包括圆（circle、solidcircle）、矩形（rectangle、solidrectangle）、扇形（pie、solidpie）、多边形（polygon、fillpolygon）等，可绘制无填充、有填充（带边框 / 无边框）的图形。

（2）文字输出：学习了 8 种文字函数，如outtextxy（指定位置输出）、drawtext（指定区域格式化输出），可通过settextstyle设置文字大小、字体，通过LOGFONT结构体自定义字体样式（斜体、下划线等）。

（3）双缓冲绘图：为解决绘图闪烁问题，学习了BeginBatchDraw（开始批量绘图）、FlushBatchDraw（刷新绘图）、EndBatchDraw（结束批量绘图）的使用，通过先在内存批量绘制再一次性输出到屏幕，提升绘图效率。

### 1.1.2 难点和解决办法

（1）坐标系统混淆：初期难以区分物理坐标与逻辑坐标，尤其是setorigin和setaspectratio的作用效果。例如，使用setorigin(300,300)后，绘图坐标会以窗口中心为原点，导致绘制的图形位置与预期不符。  
解决办法：通过对比实验理解 —— 先绘制默认坐标下的图形（如circle(100,100,50)），再修改原点后重新绘制，结合文档中的五角星绘制案例，分析setaspectratio(1,-1)如何实现 Y 轴向上为正的效果

双缓冲机制理解：直接绘图时出现严重闪烁。  
解决办法：对比开启与关闭双缓冲的效果 —— 在 “自动移动的圆” 案例中，关闭双缓冲时，圆的移动会因频繁清屏和重绘出现闪烁；开启后，通过批量处理绘图指令，闪烁消失。

### 1.1.3 学习案例

习题1绘制五子棋：

源代码：#include <graphics.h>

int main()

{

initgraph(600,600);

int step = 30;

//设置背景色为黄色

setbkcolor(RGB(255,222,173));

//用背景色清空屏幕r

cleardevice();

setlinestyle(PS\_SOLID, 2);//画实线，宽度为两个像素

setcolor(RGB(0, 0, 0));//设置为黑色

int i;

for (i = 1;i <= 19;i++)//画横线和竖线

{

int vWidth = (i == 1 || i == 19) ? 3 : 1;

setlinestyle(PS\_SOLID, vWidth);

line(i \* step, 1 \* step, i \* step, 19 \* step);

// 设置横线宽度：边缘为3，相邻为2，其余为1

int hWidth = (i == 1 || i == 19) ? 3: 1;

setlinestyle(PS\_SOLID, hWidth);

line(1 \* step, i \* step, 19 \* step, i \* step);

}

for (i = 1;i <= 19;i++)//写行数

{

TCHAR num[5];

settextcolor(YELLOW);

\_stprintf\_s(num, \_T("%d"),20-i);

if (i <= 10)

{

outtextxy(9, i \* step - 7, num);

}

else

{

outtextxy(15, i \* step - 7, num);

}

}

char ch = 'A';

for (i = 1; i <= 19; i++) {

TCHAR letter[2] = { ch++, \_T('\0') };

// 计算X坐标，使字母水平居中对齐到列线

// 每个字母的位置 = 列起始位置 + (列宽 - 字母宽度)/2

int x = i \* step - 5;

// Y坐标设置为棋盘底部下方10像素

int y = 19 \* step + 10;

outtextxy(x, y, letter);

}

setfillcolor(BLACK); // 设置填充颜色为黑色

int starPoints[5][2] = { // 星位坐标(列,行)

{4, 4}, {16, 4}, {10, 10}, {4, 16}, {16, 16}

};

for (i = 0; i < 5; i++) {

int x = starPoints[i][0] \* step;

int y = starPoints[i][1] \* step;

solidcircle(x, y, 4); // 绘制半径为4的实心圆作为星位标记

}

system("pause");

closegraph();

return 0;

}

习题2 动态时钟

源代码：#include <graphics.h>

#include <iostream>

#include <time.h>

#include <stdio.h>

#define PI 3.1415

using namespace std;

class CLK

{

public:

int x;

int y;

int hour;

int minute;

int second;

int r;

COLORREF bk\_col;// 背景色

public:

CLK(int x, int y, int h, int m, int s, int r, COLORREF bk\_color)

: x(x), y(y), hour(h), minute(m), second(s), r(r), bk\_col(bk\_color) {

}

};

class Paint

{

public:

void drawclock\_bk(CLK c); // 绘制时钟背景

void drawclock\_scale(CLK c); // 绘制刻度

void drawclock\_sharp(CLK c); // 绘制指针

void drawclock\_number(CLK c); //绘制数字

};

void Paint::drawclock\_bk(CLK c)

{

setcolor(RGB(255, 0, 255));

setfillcolor(c.bk\_col);

setlinestyle(PS\_SOLID, 5);

fillcircle(c.x, c.y, c.r);

setfillcolor(RGB(0, 0, 120)); // 中心圆点颜色

fillcircle(c.x, c.y, 5);

}

void Paint::drawclock\_scale(CLK c)

{

int x1, y1; // 刻度外端点

int x2, y2; // 刻度内端点

setlinecolor(RGB(255, 255, 255));// 刻度颜色

setlinestyle(PS\_SOLID, 4);

for (int a = 0; a < 60; a++)

{

// 计算角度（弧度）：0度在3点方向，顺时针递增

double rad = (a \* 6) \* PI / 180; // 每格6度（360/60）

// 外端点（表盘边缘）

x1 = static\_cast<int>(c.x + cos(rad) \* c.r);

y1 = static\_cast<int>(c.y - sin(rad) \* c.r); // y轴向下为正，所以减sin

// 内端点（刻度长度：整点刻度更长）

if (a % 5 == 0) // 整点刻度（12个）

{

x2 = static\_cast<int>(c.x + cos(rad) \* (c.r - 25));

y2 = static\_cast<int>(c.y - sin(rad) \* (c.r - 25));

}

else // 非整点刻度（48个）

{

x2 = static\_cast<int>(c.x + cos(rad) \* (c.r - 15));

y2 = static\_cast<int>(c.y - sin(rad) \* (c.r - 15));

}

line(x1, y1, x2, y2);

}

}

void Paint::drawclock\_sharp(CLK c)

{

int x\_hour, y\_hour;

int x\_min, y\_min;

int x\_sec, y\_sec;

double sec\_rad = (60 - c.second) \* 6 \* PI / 180;

double min\_rad = (60 - (c.minute + c.second / 60.0)) \* 6 \* PI / 180;

double hour\_rad = (12 - (c.hour % 12 + c.minute / 60.0)) \* 30 \* PI / 180;

x\_hour = static\_cast<int>(c.x + cos(hour\_rad) \* c.r \* 0.4);

y\_hour = static\_cast<int>(c.y - sin(hour\_rad) \* c.r \* 0.4);

x\_min = static\_cast<int>(c.x + cos(min\_rad) \* c.r \* 0.6);

y\_min = static\_cast<int>(c.y - sin(min\_rad) \* c.r \* 0.6);

x\_sec = static\_cast<int>(c.x + cos(sec\_rad) \* c.r \* 0.8);

y\_sec = static\_cast<int>(c.y - sin(sec\_rad) \* c.r \* 0.8);

setlinecolor(RGB(0, 0, 255)); // 时针蓝色

setlinestyle(PS\_SOLID, 5); // 时针粗

line(c.x, c.y, x\_hour, y\_hour);

setlinecolor(RGB(0, 255, 0)); // 分针绿色

setlinestyle(PS\_SOLID, 3); // 分针中等粗细

line(c.x, c.y, x\_min, y\_min);

setlinecolor(RGB(255, 0, 0)); // 秒针红色

setlinestyle(PS\_SOLID, 1); // 秒针细

line(c.x, c.y, x\_sec, y\_sec);

}

void Paint::drawclock\_number(CLK c)

{

int x\_num, y\_num;

setcolor(RGB(0, 0, 125));

setfillcolor(RGB(0, 0, 125));

for (int a = 0; a < 60; a += 5)

{

double rad = (90 - a \* 6) \* PI / 180;

x\_num = static\_cast<int>(c.x + cos(rad) \* (c.r - 30));

y\_num = static\_cast<int>(c.y - sin(rad) \* (c.r - 30));

int displayNum = (a / 5 == 0) ? 12 : a / 5;

TCHAR num[5];

// 设置文字样式

settextcolor(YELLOW);

settextstyle(30, 10, \_T("Arial"));

\_stprintf\_s(num, \_T("%d"), displayNum);

// 计算文字宽高

RECT rect = { 0, 0, 100, 100 };

drawtext(num, &rect, DT\_CALCRECT);

int textWidth = rect.right - rect.left;

int textHeight = rect.bottom - rect.top;

// 设置并绘制文字背景（与表盘同色）

setfillcolor(c.bk\_col);

solidrectangle(x\_num - textWidth / 2 - 2,

y\_num - textHeight / 2 - 2,

x\_num + textWidth / 2 + 2,

y\_num + textHeight / 2 + 2);

// 绘制文字

outtextxy(x\_num - textWidth / 2, y\_num - textHeight / 2, num);

}

}

int main()

{

initgraph(1024, 800);

setbkcolor(RGB(0, 0, 0));

cleardevice();

time\_t nowtime;

struct tm timeinfo;

time(&nowtime);

localtime\_s(&timeinfo, &nowtime);

CLK clock(512, 288, timeinfo.tm\_hour, timeinfo.tm\_min, timeinfo.tm\_sec, 200, RGB(0, 0, 0));

Paint painter;

while (true)

{

cleardevice();

time\_t nowtime;

struct tm timeinfo;

time(&nowtime);

localtime\_s(&timeinfo, &nowtime);

// 更新时钟对象

clock.hour = timeinfo.tm\_hour;

clock.minute = timeinfo.tm\_min;

clock.second = timeinfo.tm\_sec;

painter.drawclock\_bk(clock); // 绘制背景

painter.drawclock\_scale(clock); // 绘制刻度

painter.drawclock\_sharp(clock); // 绘制指针

painter.drawclock\_number(clock); // 绘制数字

Sleep(1000);

}

// 关闭图形窗口

closegraph();

return 0;

}

## 1.2 第二周学习总结（同上）

### 1.2.1 内容简介

本周学习了 EasyX 的进阶知识，聚焦 “图像处理 - 消息交互 - 动画与多媒体”，具体包括：

（1）图像处理：

核心函数：掌握loadimage（从文件加载图像到IMAGE对象或窗口）、putimage（绘制图像，支持截取部分区域和三元光栅操作）、resize（调整图像尺寸）、rotateimage（旋转图像）等函数，可实现图像的加载、显示、缩放和旋转。

透明贴图：学习了 3 种实现方式 —— 通过掩码图与前景图的三元光栅操作（SRCAND和SRCPAINT）、TransparentBlt函数（支持指定透明色）、AlphaBlend函数（支持 Alpha 通道和半透明），解决图像叠加时的背景遮挡问题。

IMAGE类：理解IMAGE对象的属性（宽高通过getwidth/getheight获取）和方法（支持直接赋值、缓冲区操作），可通过GetImageBuffer获取像素缓冲区直接修改图像像素。

（2）消息处理：

消息函数：学习peekmessage（非阻塞获取消息）和getmessage（阻塞获取消息），通过ExMessage结构体解析鼠标（移动、点击）、键盘（按键按下 / 松开）、窗口消息。

交互实现：掌握鼠标消息处理（如WM\_MOUSEMOVE实现跟随鼠标的图形）、键盘消息处理（如WM\_KEYDOWN/WM\_KEYUP实现方向键控制物体移动），支持组合键（如Ctrl+鼠标键）和斜向移动（通过向量标准化计算坐标增量）。

（3）动画与多媒体：

动画帧控制：通过GetTickCount记录时间，控制动画帧间隔（如每 15ms 切换一帧），实现角色动画（循环播放序列图像）。

窗口与多媒体扩展：学习设置窗口标题（SetWindowText）、弹窗消息（MessageBox）、播放音频（mciSendString控制音频文件的打开、播放、循环）。

### 1.2.2 难点和解决办法

（1）消息循环逻辑混乱：在 “键盘控制小球” 案例中，初期使用WM\_KEYDOWN直接移动小球，导致斜向移动时速度异常（对角线速度快于单方向）。  
解决办法：改用 “状态标记法”—— 定义isMoveUp等布尔变量记录按键状态，在消息处理中更新状态，在主循环中根据状态计算移动向量，通过向量标准化（factorX=directX/directXY）使斜向移动速度与单方向一致

（2）动画帧同步问题：实现角色动画时，因电脑性能差异，动画播放速度不稳定  
解决办法：通过GetTickCount计算每帧耗时，若耗时小于目标间隔（如 1000/60 ms），则用Sleep补时，确保每秒播放 60 帧，使动画在不同设备上保持一致速度

### 1.2.3 学习案例

习题3 鼠标加键盘绘图：  
源代码：#include<graphics.h>

#include<iostream>

using namespace std;

COLORREF currentColor = RGB(255, 255, 255); // 默认黑色

const int NORMAL\_SIZE = 10; // 普通尺寸

const int LARGE\_SIZE = 20; // 放大尺寸

void drawSquare(int x,int y,int size)

{

int half = size / 2;

setcolor(currentColor);

setfillcolor(currentColor);

solidrectangle(x - half, y - half, x + half, y + half);

}

void drawCircle(int x, int y, int r)

{

setcolor(currentColor);

setfillcolor(currentColor);

solidcircle(x, y, r);

}

void showInstructions() {

settextcolor(BLACK);

// 使用\_T宏将字符串转为宽字符，匹配outtextxy的参数要求

outtextxy(10, 10, \_T("左键: 正方形 | 右键: 圆形"));

outtextxy(10, 30, \_T("Ctrl+鼠标键: 放大图形"));

outtextxy(10, 50, \_T("C:清屏 | R/G/B/W:切换颜色 | ESC:退出"));

}

bool isCtrlPressed()

{

// GetKeyState返回按键状态，最高位为1表示按下

return (GetKeyState(VK\_CONTROL) & 0x8000) != 0;

}

int main()

{

bool running = true; // 主循环控制参数

ExMessage msg; // 消息对象

initgraph(800, 600); // 初始化绘图窗口

BeginBatchDraw(); // 开启批量绘图

showInstructions();

// 主循环

while (running) {

// 消息处理

if (peekmessage(&msg, EM\_MOUSE | EM\_KEY))

{

bool ctrlPressed = isCtrlPressed();

int currentSize = ctrlPressed ? LARGE\_SIZE : NORMAL\_SIZE;

if (msg.message == WM\_LBUTTONDOWN)

{

drawSquare(msg.x, msg.y, currentSize);

}

else if (msg.message == WM\_RBUTTONDOWN)

{

drawCircle(msg.x, msg.y, currentSize);

}

else if (msg.message == WM\_KEYDOWN)

{

switch (msg.vkcode)

{

case 'C': case 'c': // 清屏

cleardevice();

// 重绘提示文字

settextcolor(BLACK);

break;

case 'R': case 'r': currentColor = RGB(255, 0, 0); break; // 红色

case 'G': case 'g': currentColor = RGB(0, 255, 0); break; // 绿色

case 'B': case 'b': currentColor = RGB(0, 0, 255); break; // 蓝色

case 'W': case 'w': currentColor = RGB(255, 255, 255); break; // 白色

case VK\_ESCAPE: // 退出程序

running = false; // 退出循环

break;

}

}

}

cleardevice;

FlushBatchDraw(); // 刷新批量绘图

}

EndBatchDraw(); // 关闭批量绘图

closegraph(); // 关闭绘图窗口

return 0;

}

习题4弹球

源代码：#include<graphics.h>

#include<iostream>

using namespace std;

COLORREF currentColor = RGB(255, 255, 255); // 默认黑色

const int NORMAL\_SIZE = 10; // 普通尺寸

const int LARGE\_SIZE = 20; // 放大尺寸

void drawSquare(int x,int y,int size)

{

int half = size / 2;

setcolor(currentColor);

setfillcolor(currentColor);

solidrectangle(x - half, y - half, x + half, y + half);

}

void drawCircle(int x, int y, int r)

{

setcolor(currentColor);

setfillcolor(currentColor);

solidcircle(x, y, r);

}

void showInstructions() {

settextcolor(BLACK);

// 使用\_T宏将字符串转为宽字符，匹配outtextxy的参数要求

outtextxy(10, 10, \_T("左键: 正方形 | 右键: 圆形"));

outtextxy(10, 30, \_T("Ctrl+鼠标键: 放大图形"));

outtextxy(10, 50, \_T("C:清屏 | R/G/B/W:切换颜色 | ESC:退出"));

}

bool isCtrlPressed()

{

// GetKeyState返回按键状态，最高位为1表示按下

return (GetKeyState(VK\_CONTROL) & 0x8000) != 0;

}

int main()

{

bool running = true; // 主循环控制参数

ExMessage msg; // 消息对象

initgraph(800, 600); // 初始化绘图窗口

BeginBatchDraw(); // 开启批量绘图

showInstructions();

// 主循环

while (running) {

// 消息处理

if (peekmessage(&msg, EM\_MOUSE | EM\_KEY))

{

bool ctrlPressed = isCtrlPressed();

int currentSize = ctrlPressed ? LARGE\_SIZE : NORMAL\_SIZE;

if (msg.message == WM\_LBUTTONDOWN)

{

drawSquare(msg.x, msg.y, currentSize);

}

else if (msg.message == WM\_RBUTTONDOWN)

{

drawCircle(msg.x, msg.y, currentSize);

}

else if (msg.message == WM\_KEYDOWN)

{

switch (msg.vkcode)

{

case 'C': case 'c': // 清屏

cleardevice();

// 重绘提示文字

settextcolor(BLACK);

break;

case 'R': case 'r': currentColor = RGB(255, 0, 0); break; // 红色

case 'G': case 'g': currentColor = RGB(0, 255, 0); break; // 绿色

case 'B': case 'b': currentColor = RGB(0, 0, 255); break; // 蓝色

case 'W': case 'w': currentColor = RGB(255, 255, 255); break; // 白色

case VK\_ESCAPE: // 退出程序

running = false; // 退出循环

break;

}

}

}

cleardevice;

FlushBatchDraw(); // 刷新批量绘图

}

EndBatchDraw(); // 关闭批量绘图

closegraph(); // 关闭绘图窗口

return 0;

}

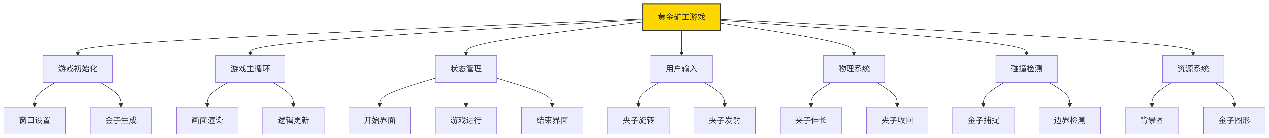
# 2 项目开发报告

## 2.1 项目简介

本项目基于 EasyX 图形库开发的 "黄金矿工" 模仿游戏，基本实现了经典黄金矿工游戏的核心玩法。玩家通过控制矿工发射和回收吊钩，抓取不同价值的黄金，在限定时间内获取足够分数以通关。

## 2.2 需求分析

### 2.2.1 功能模块



### 2.2.2 需求说明

**（1）界面渲染需求**

背景图需适配游戏窗口尺寸（800×600 像素），加载失败时显示深色调网格背景，避免空白。

矿工与吊钩需可视化呈现：矿工固定在顶部中间，吊钩随角度旋转，伸长 / 收回过程流畅无卡顿。

黄金需区分大小（小 / 中 / 大），通过半径差异（8/12/16 像素）和光泽效果（白色高光点）直观区分。

文本信息需清晰可见：分数和时间分别显示在左上角和右上角，游戏状态文字需突出显示

**（2）输入控制需求**

旋转控制：Q 键使吊钩逆时针旋转，E 键使吊钩顺时针旋转，旋转速度固定（0.8 度 / 帧），仅在吊钩未发射时生效。

发射控制：空格键触发吊钩发射，发射后吊钩自动伸长，达到最大长度（550 像素）后自动收回。

状态切换：Enter 键在开始界面触发游戏启动，在游戏结束界面触发重新开始；所有按键响应需无延迟，支持连续输入（如长按 Q/E 持续旋转）。

**（3）游戏逻辑需求**

吊钩运动：伸长速度固定为 5 像素 / 帧，收回速度根据黄金大小调整（小黄金 6 像素 / 帧，中黄金 4 像素 / 帧，大黄金 2 像素 / 帧）。

碰撞判定：吊钩末端与黄金的距离小于碰撞范围（小 15 / 中 20 / 大 25 像素）时，判定为抓取成功，黄金随吊钩移动。

分数规则：小黄金得 5 分，中黄金得 10 分，大黄金得 20 分，累计分数达到 100 分即判定胜利。

时间规则：游戏开始后倒计时 60 秒，时间为 0 时结束游戏；若提前达到目标分数，立即结束并判定胜利。

**（4）状态管理需求**

开始界面：显示游戏标题 “黄金矿工”、操作说明（Q/E 旋转、空格发射）和目标（60 秒 100 分），等待 Enter 键触发。

游戏中：实时更新吊钩状态、黄金位置、分数和时间，检测胜利 / 失败条件（分数达标或时间结束）。

游戏结束：显示 “恭喜胜利” 或 “游戏结束” 提示，显示最终分数，等待 Enter 键返回开始界面。

## 2.3 系统设计

### 2.3.1 界面设计（UI）

图片包含 图表

AI 生成的内容可能不正确。

### 2.3.2 流程设计

图形用户界面

AI 生成的内容可能不正确。

图2.3 系统流程图

### 2.3.3 类设计（可选）

如果使用C++面向对象开发，则此部分可以绘制单个类图或类关系图，例如：

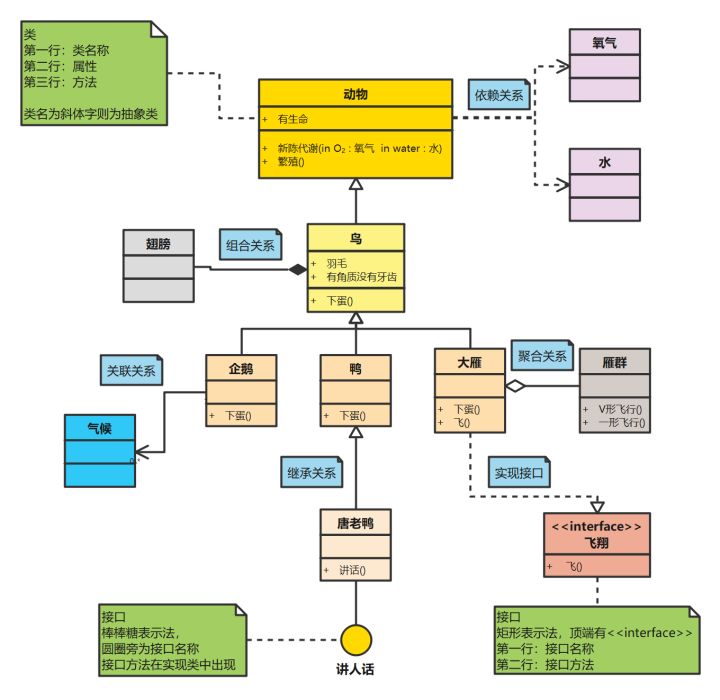


图2.4 类关系图

## 2.4 系统实现

### 2.4.1 实现效果

文本

AI 生成的内容可能不正确。

图1.开始界面图片包含 背景图案

AI 生成的内容可能不正确。

图2.抓取黄金图片包含 图形用户界面

AI 生成的内容可能不正确。

图3.碰到边界收回

### 2.4.2 核心代码

#include <graphics.h>

#include <windows.h>

#include <mmsystem.h>

#include <ctime>

#include <cmath>

#include <cwchar>

#include <vector>

#pragma comment(lib, "winmm.lib")

const int WIDTH = 800;

const int HEIGHT = 600;

const int FPS = 60;

const int FRAME\_DELAY = 1000 / FPS;

const int TARGET\_SCORE = 100;

const int COUNTDOWN\_SECONDS = 60;

// 游戏状态枚举：定义游戏的不同阶段

enum GameState { START\_SCREEN, PLAYING, GAME\_OVER };

GameState currentState = START\_SCREEN;

bool isPaused = false;

// 大小枚举：定义不同尺寸的黄金

enum GoldSize { SMALL, MEDIUM, LARGE };

// 黄金结构体

struct Gold {

int x, y;

GoldSize size;

int value;

// 构造函数

Gold(int \_x, int \_y) {

x = \_x;

y = \_y;

// 随机生成黄金大小（0-2的随机数）

int r = rand() % 3;

if (r == 0) {

size = SMALL;

value = 5;

}

else if (r == 1) {

size = MEDIUM;

value = 10;

}

else {

size = LARGE;

value = 20;

}

}

};

// 游戏元素变量定义

int minerX = WIDTH / 2;

int minerY = HEIGHT - 500;

int clawBaseY = minerY + 5;

int clawLength = 600;

int currentClawLength = 0;

double clawAngle = 0;

bool isClawActive = false;

bool isRetracting = false;

int score = 0;

int carriedGoldIndex = -1;

DWORD gameStartTime = 0;

std::vector<Gold> golds;

IMAGE backgroundImg;

// 角度转换常量：度转弧度（用于三角函数计算）

const double DEG\_TO\_RAD = 3.141592653589793 / 180.0;

// 初始化函数：设置游戏初始环境

void setup() {

initgraph(WIDTH, HEIGHT);

srand((unsigned int)time(0));

loadimage(&backgroundImg, L"background.jpg", WIDTH, HEIGHT, true);

// 生成初始的5个随机位置的黄金

for (int i = 0; i < 5; ++i) {

int x = rand() % (WIDTH - 100) + 50;

int y = rand() % (HEIGHT - 100) + 100;

golds.push\_back(Gold(x, y));

}

}

// 绘制开始界面：显示游戏标题和操作说明

void drawStartScreen() {

setbkcolor(BLACK);

cleardevice();

// 绘制游戏标题

settextcolor(YELLOW);

settextstyle(50, 0, L"黑体");

outtextxy(WIDTH / 2 - 150, HEIGHT / 2 - 100, L"黄金矿工");

// 绘制操作说明和游戏目标

settextcolor(WHITE);

settextstyle(25, 0, L"宋体");

outtextxy(WIDTH / 2 - 180, HEIGHT / 2 + 50, L"按Enter键开始游戏");

outtextxy(WIDTH / 2 - 200, HEIGHT / 2 + 80, L"Q/E键旋转夹子 空格键发射");

outtextxy(WIDTH / 2 - 180, HEIGHT / 2 + 110, L"目标: 60秒内获得100分");

outtextxy(WIDTH / 2 - 150, HEIGHT / 2 + 140, L"P键: 暂停/继续游戏");

}

// 绘制背景：显示背景图片和分隔线

void drawBackground() {

putimage(0, 0, &backgroundImg);

setlinecolor(RGB(100, 100, 100));

line(0, 100, WIDTH, 100);

}

// 绘制矿工：绘制矿工主体和支架

void drawMiner() {

setfillcolor(YELLOW);

solidrectangle(minerX - 15, minerY - 20, minerX + 15, minerY - 5);

// 绘制支架（棕色矩形，连接矿工和夹子）

setfillcolor(BROWN);

solidrectangle(minerX - 5, minerY - 4, minerX + 5, minerY + 5);

}

// 绘制夹子

void drawClaw() {

// 确定夹子显示长度

int displayLength = isClawActive ? currentClawLength : 20;

// 计算夹子末端坐标

int clawEndX = minerX + displayLength \* cos(clawAngle);

int clawEndY = clawBaseY + displayLength \* sin(clawAngle);

// 绘制夹子绳索

setlinecolor(LIGHTBLUE);

setlinestyle(PS\_SOLID, 6);

line(minerX, clawBaseY, clawEndX, clawEndY);

// 夹子主体参数

const int CLAW\_WIDTH = 20;

const int CLAW\_HEIGHT = 25;

const int TOOTH\_WIDTH = 4;

const int TOOTH\_LENGTH = 10;

// 计算角度的正弦和余弦值

double sinA = sin(clawAngle);

double cosA = cos(clawAngle);

// 定义夹子主体的四个顶点

POINT clawBody[4] = {

{ (int)(clawEndX - CLAW\_WIDTH / 2 \* sinA - CLAW\_HEIGHT \* cosA),

(int)(clawEndY + CLAW\_WIDTH / 2 \* cosA - CLAW\_HEIGHT \* sinA) },

{ (int)(clawEndX + CLAW\_WIDTH / 2 \* sinA - CLAW\_HEIGHT \* cosA),

(int)(clawEndY - CLAW\_WIDTH / 2 \* cosA - CLAW\_HEIGHT \* sinA) },

{ (int)(clawEndX + CLAW\_WIDTH / 2 \* sinA),

(int)(clawEndY - CLAW\_WIDTH / 2 \* cosA) },

{ (int)(clawEndX - CLAW\_WIDTH / 2 \* sinA),

(int)(clawEndY + CLAW\_WIDTH / 2 \* cosA) }

};

// 绘制夹子主体

setfillcolor(BLUE);

fillpolygon(clawBody, 4);

// 定义左爪齿的三个顶点

POINT leftTooth[3] = {

clawBody[2],

{ (int)(clawBody[2].x + TOOTH\_LENGTH \* cosA + TOOTH\_WIDTH \* sinA),

(int)(clawBody[2].y + TOOTH\_LENGTH \* sinA - TOOTH\_WIDTH \* cosA) },

{ (int)(clawBody[2].x + TOOTH\_LENGTH \* cosA - TOOTH\_WIDTH \* sinA),

(int)(clawBody[2].y + TOOTH\_LENGTH \* sinA + TOOTH\_WIDTH \* cosA) }

};

setfillcolor(WHITE);

fillpolygon(leftTooth, 3);

// 定义右爪齿的三个顶点

POINT rightTooth[3] = {

clawBody[3],

{ (int)(clawBody[3].x + TOOTH\_LENGTH \* cosA + TOOTH\_WIDTH \* sinA),

(int)(clawBody[3].y + TOOTH\_LENGTH \* sinA - TOOTH\_WIDTH \* cosA) },

{ (int)(clawBody[3].x + TOOTH\_LENGTH \* cosA - TOOTH\_WIDTH \* sinA),

(int)(clawBody[3].y + TOOTH\_LENGTH \* sinA + TOOTH\_WIDTH \* cosA) }

};

setfillcolor(WHITE);

fillpolygon(rightTooth, 3);

}

// 绘制所有黄金

void drawGolden() {

// 遍历所有黄金并绘制

for (const auto& gold : golds) {

// 根据黄金大小设置半径

int radius;

switch (gold.size) {

case SMALL: radius = 8; break;

case MEDIUM: radius = 12; break;

case LARGE: radius = 16; break;

}

// 绘制黄金主体

setfillcolor(YELLOW);

solidcircle(gold.x, gold.y, radius);

// 绘制光泽效果

setfillcolor(WHITE);

solidcircle(gold.x - radius / 3, gold.y - radius / 3, radius / 5);

}

}

// 绘制分数：显示当前分数和目标分数

void drawScore() {

wchar\_t scoreText[50];

swprintf(scoreText, 50, L"分数: %d / %d", score, TARGET\_SCORE);

settextcolor(WHITE);

settextstyle(20, 0, L"宋体");

outtextxy(10, 10, scoreText);

}

// 绘制计时器：显示剩余时间，暂停时不更新

void drawTimer() {

if (isPaused) return;

DWORD elapsedTime = (timeGetTime() - gameStartTime) / 1000;

int remainingTime = COUNTDOWN\_SECONDS - elapsedTime;

// 时间到则结束游戏

if (remainingTime <= 0) {

remainingTime = 0;

currentState = GAME\_OVER;

}

// 格式化时间文本

wchar\_t timerText[20];

swprintf(timerText, 20, L"时间: %02d秒", remainingTime);

settextcolor(WHITE);

settextstyle(20, 0, L"宋体");

outtextxy(WIDTH - 130, 10, timerText);

}

// 绘制游戏结束或暂停界面：显示遮罩、分数和操作提示

void drawGameOver() {

// 绘制半透明黑色遮罩

setfillcolor(RGB(0, 0, 0, 180));

solidrectangle(0, 0, WIDTH, HEIGHT);

// 绘制标题文字

settextcolor(YELLOW);

settextstyle(50, 0, L"黑体");

// 根据状态显示不同标题

if (currentState == GAME\_OVER) {

bool gameWon = (score >= TARGET\_SCORE);

if (gameWon) {

outtextxy(WIDTH / 2 - 150, HEIGHT / 2 - 80, L"恭喜胜利!");

} else {

outtextxy(WIDTH / 2 - 150, HEIGHT / 2 - 80, L"游戏结束");

}

} else if (isPaused) {

outtextxy(WIDTH / 2 - 120, HEIGHT / 2 - 80, L"游戏暂停");

}

// 显示当前分数

settextcolor(WHITE);

settextstyle(30, 0, L"宋体");

wchar\_t finalScore[50];

swprintf(finalScore, 50, L"当前分数: %d", score);

outtextxy(WIDTH / 2 - 100, HEIGHT / 2 + 20, finalScore);

// 显示操作提示

settextstyle(20, 0, L"宋体");

if (currentState == GAME\_OVER) {

outtextxy(WIDTH / 2 - 120, HEIGHT / 2 + 80, L"按Enter键重新开始");

} else if (isPaused) {

outtextxy(WIDTH / 2 - 120, HEIGHT / 2 + 80, L"按P键继续游戏");

}

}

// 检测按键是否按下

bool isKeyPressed(int key) {

// GetAsyncKeyState返回按键状态，&0x8000判断是否按下

return GetAsyncKeyState(key) & 0x8000;

}

// 重置游戏：将所有游戏变量恢复到初始状态

void resetGame() {

score = 0;

isClawActive = false;

isRetracting = false;

currentClawLength = 0;

clawAngle = 0;

carriedGoldIndex = -1;

golds.clear();

isPaused = false;

// 重新生成5个黄金

for (int i = 0; i < 5; ++i) {

int x = rand() % (WIDTH - 100) + 50;

int y = rand() % (HEIGHT - 100) + 100;

golds.push\_back(Gold(x, y));

}

gameStartTime = timeGetTime();

}

// 移动控制：处理键盘输入，控制夹子旋转和发射

void moveMiner() {

// 非游戏状态或暂停时不响应控制

if (currentState != PLAYING || isPaused) return;

// 夹子未激活时（未发射），可旋转

if (!isClawActive) {

if (isKeyPressed('Q')) {

clawAngle += 0.8 \* DEG\_TO\_RAD;

}

if (isKeyPressed('E')) {

clawAngle -= 0.8 \* DEG\_TO\_RAD;

}

}

// 空格键发射夹子（仅当夹子未激活时）

if (isKeyPressed(VK\_SPACE) && !isClawActive) {

isClawActive = true;

isRetracting = false;

currentClawLength = 0;

carriedGoldIndex = -1;

}

}

// 更新游戏逻辑：处理夹子移动、碰撞检测和分数更新

void updateGame() {

if (currentState != PLAYING || isPaused) return;

if (!isClawActive) return;

// 更新夹子长度

if (!isRetracting) {

// 夹子伸长阶段：长度增加5像素/帧

currentClawLength += 5;

// 达到最大长度后开始收回

if (currentClawLength >= clawLength) {

isRetracting = true;

}

} else {

// 夹子收回阶段：根据是否携带黄金调整速度

int retractSpeed = 5;

if (carriedGoldIndex != -1) {

// 携带不同大小的黄金，收回速度不同

switch (golds[carriedGoldIndex].size) {

case SMALL: retractSpeed = 6; break;

case MEDIUM: retractSpeed = 4; break;

case LARGE: retractSpeed = 2; break;

}

}

currentClawLength -= retractSpeed;

// 夹子收回至起点时，结束激活状态

if (currentClawLength <= 0) {

isClawActive = false;

// 若携带黄金，则加分并重新生成该黄金

if (carriedGoldIndex != -1) {

score += golds[carriedGoldIndex].value;

// 达到目标分数则游戏结束

if (score >= TARGET\_SCORE) {

currentState = GAME\_OVER;

}

// 重新生成黄金（随机位置）

golds[carriedGoldIndex] = Gold(

rand() % (WIDTH - 100) + 50,

rand() % (HEIGHT - 100) + 100

);

carriedGoldIndex = -1;

}

}

}

// 计算夹子末端当前坐标

int clawEndX = minerX + currentClawLength \* cos(clawAngle);

int clawEndY = clawBaseY + currentClawLength \* sin(clawAngle);

// 若携带黄金，更新黄金位置（跟随夹子末端）

if (carriedGoldIndex != -1) {

golds[carriedGoldIndex].x = clawEndX;

golds[carriedGoldIndex].y = clawEndY;

} else {

// 碰撞检测：检查夹子末端是否碰到黄金

for (int i = 0; i < golds.size(); ++i) {

// 根据黄金大小设置碰撞检测范围

int collisionRange;

switch (golds[i].size) {

case SMALL: collisionRange = 15; break;

case MEDIUM: collisionRange = 20; break;

case LARGE: collisionRange = 25; break;

}

// 计算夹子末端与黄金的距离

double distance = sqrt(pow(clawEndX - golds[i].x, 2) + pow(clawEndY - golds[i].y, 2));

// 距离小于碰撞范围，则夹住黄金

if (distance < collisionRange) {

carriedGoldIndex = i; // 记录携带的黄金索引

isRetracting = true; // 开始收回

break;

}

}

}

// 边界检测：夹子超出屏幕范围则开始收回

if (clawEndX < 0 || clawEndX > WIDTH || clawEndY < 0 || clawEndY > HEIGHT) {

isRetracting = true;

}

}

// 游戏主循环：控制游戏流程、帧率和状态切换

void gameLoop() {

DWORD lastFrameTime = timeGetTime(); // 上一帧的时间

// 无限循环，持续运行游戏

while (true) {

DWORD currentTime = timeGetTime(); // 当前时间

DWORD elapsedTime = currentTime - lastFrameTime;

// 控制帧率：当时间差大于等于帧间隔时，更新游戏

if (elapsedTime >= FRAME\_DELAY) {

lastFrameTime = currentTime; // 更新上一帧时间

// 检测暂停按键

if (currentState == PLAYING && isKeyPressed('P')) {

isPaused = !isPaused;

Sleep(200);

}

BeginBatchDraw();

// 根据当前游戏状态执行不同操作

if (currentState == START\_SCREEN) {

drawStartScreen();

// 按Enter键进入游戏

if (isKeyPressed(VK\_RETURN)) {

currentState = PLAYING;

resetGame();

}

} else if (currentState == PLAYING) {

drawBackground();

moveMiner();

updateGame();

drawMiner();

drawClaw();

drawGolden();

drawScore();

drawTimer();

// 若暂停，绘制暂停界面

if (isPaused) {

drawGameOver(); // 复用结束界面的遮罩效果

}

} else if (currentState == GAME\_OVER) {

drawGameOver(); // 绘制结束界面

// 按Enter键返回开始界面

if (isKeyPressed(VK\_RETURN)) {

currentState = START\_SCREEN;

}

}

FlushBatchDraw(); // 执行批量绘图，显示所有内容

} else {

// 未达到帧间隔，休眠等待

Sleep(FRAME\_DELAY - elapsedTime);

}

}

}

// 主函数：程序入口

int main() {

setup(); // 初始化游戏

gameLoop(); // 运行游戏主循环

closegraph(); // 关闭图形窗口

return 0;

## }2.5 系统测试

此部分为系统测试用例表，写10个以上测试用例，重点针对错误值和边界值进行测试，例如：

表2.1 系统测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 用例编号 | 测试项 | 测试步骤 | 输入信号 | 预期结果 | 测试结果 |
| 1 | 开始游戏 | 按下键盘的“Enter”键 | 键盘 | 进入游戏，显示游戏界面 | 通过 |
| 2 | 夹子逆时针旋转 | 按下键盘的“q”键 | 键盘 | 夹子会逆时针旋转 | 通过 |
| 3 | 夹子顺时针旋转 | 按下键盘的“q”键 | 键盘 | 夹子会顺时针旋转 | 通过 |
| 4 | 夹子发射 | 按下键盘的“Space”键 | 键盘 | 夹子能发射出去 | 通过 |
| 5 | 夹子碰到边界会自动收回 | 按下键盘的“Space”键后，夹子朝着边界射出 | 键盘 | 夹子碰到边界会自动回收 | 通过 |
| 6 | 夹子在发射过程中不会旋转 | 按下键盘的“Space”键后，再按下“q”或“e”键 | 键盘 | 夹子发射过程中不会因为按下“q”或“e”键而旋转 | 通过 |
| 7 | 夹子碰到黄金会携带黄金回收 | 按下键盘的“Space”键后，朝着黄金方向射出，碰到黄金后，判定碰撞，携带黄金回收 | 键盘 | 夹子能识别碰到黄金，并携带黄金收回 | 通过 |
| 8 | 游戏暂停 | 按下键盘的“p”键 | 键盘 | 游戏暂停 | 通过 |
| 9 | 游戏继续 | 按下键盘的“p”键 | 键盘 | 游戏继续 | 通过 |
| 10 | 游戏结束后重新开始 | 按下键盘的“Enter”键 | 键盘 | 游戏重新开始 | 通过 |

## 2.6 项目总结

一、总体完成情况

本黄金矿工模仿游戏项目已实现核心玩法和基础功能，包括游戏流程控制，核心交互，计分与计时系统、碰撞检测及双缓冲绘图优化。

（1）完整的游戏流程：通过状态机管理START\_SCREEN、PLAYING、GAME\_OVER三种状态，支持按 Enter 键开始 / 重启游戏，逻辑清晰。

（2）核心玩法实现：玩家可通过 Q/E 键旋转夹子、空格键发射，夹子会自动抓取黄金并根据黄金大小调整回收速度，抓取后自动刷新黄金位置，计分规则符合预期。

（3）用户体验优化：添加暂停功能（P 键控制），暂停时冻结游戏逻辑与倒计时；采用双缓冲绘图（BeginBatchDraw/FlushBatchDraw）消除画面闪烁，提升流畅度。

（4）容错处理：背景图加载失败时不影响游戏运行，通过合理的坐标计算确保元素不超出窗口边界。

二、技术难点与解决方法

（1）夹子旋转与碰撞检测

难点：夹子旋转时需实时计算末端坐标，且不同大小的黄金碰撞范围需精准匹配。

解决：利用三角函数（cos/sin）计算旋转后的坐标，根据黄金半径动态设置碰撞范围（小 15px / 中 20px / 大 25px），通过距离公式判断是否抓取。

（2）双缓冲绘图与闪烁问题

难点：频繁刷新屏幕导致夹子移动时出现闪烁。

解决：采用 EasyX 的批量绘图函数，所有绘制操作先在内存完成，再通过FlushBatchDraw一次性输出到屏幕，有效消除闪烁。

（3）暂停功能的状态管理

难点：暂停时需冻结计时、夹子运动等逻辑，同时保留当前游戏状态。

解决：新增isPaused变量标记暂停状态，在updateGame、drawTimer等函数中添加状态判断，暂停时停止逻辑更新，仅保留画面显示。

三、不足与改进方向

（1）功能单一：当前仅包含黄金一种道具，可增加岩石（扣分）、钻石（高分）等元素，丰富游戏策略性。

（2）视觉效果简单：矿工、夹子等元素为基础几何图形，可替换为图片资源提升视觉体验。

（3）交互反馈不足：操作时无音效（如夹子发射、黄金抓取音效），

四、总结

本项目通过 EasyX 图形库实现了黄金矿工的核心玩法，掌握了 2D 游戏开发的基础流程（状态管理、碰撞检测、绘图优化等）。