# 高级语言程序设计大作业实验报告

## 一、作业题目

**基于《帝国时代1》改编的即时战略游戏AI设计与实现**

### 作业简介：

本次作模拟了即时战略游戏搭建的博弈系统，该博弈系统是基于《**帝国时代1**》（微软1997年发行的即时战略游戏）改编的Qt项目。原游戏从模拟人类文明出现的石器时代开始进行，需要玩家搜集资源以生产各种单位及建筑物。当玩家搜集足够的资源及建筑特定的建筑物之后，就可以升级至下一个文明时代。本系统2.0版本复刻了原游戏石器时代和工具时代的大部分内容和操作。

### 核心目标：

- 使用Qt框架开发博弈系统，实现通过程序接口控制游戏流程。
- 实现资源采集、建筑建造、科技升级、军队训练及防御敌人进攻等功能。
- 在25分钟内击败敌人，保护己方市镇中心不被摧毁。

## 二、开发软件

1. **开发环境：**
   - **Qt Creator**：用于搭建游戏框架及可视化界面。
   - **C++编译器**：GCC/MSVC，支持C++17及以上标准。
   - **调试工具**：Qt内置调试器、`DebugText`接口输出调试信息。
2. **依赖库：**
   - Qt核心库（GUI、多线程、数据结构）。
   - STL容器（`vector`、`map`）用于管理游戏对象。
3. **辅助工具：**
   - 版本控制：Git，管理代码迭代。
   - 性能优化：Release模式编译以提高运行效率。

## 三、课题要求

学生自选题目，使用C++语言完成一个图形化的小程序。

图形化平台不限，可以是MFC、QT等任何C++图形化平台(不限制)。

程序内容主题不限，可以是小游戏、小工具等。

### 提交事项：

#### 1.在Gitee或者github上的项目网址。

将大作业代码提交到github或者gitee（码云，中国开源社区平台，对标github）www.gitee.com，要求能够体现大作业完成过程中的不同版本迭代。在创建项目时，点击页面顶部菜单栏账号左侧的加号（位于页面右侧）来新建仓库。创建仓库时，应选择公开方式，C++语言。

**注意：**

（a） 建议在初建项目时即向gitee/github提交代码，要求向gitee/github的提交能够体现大作业完成过程，**向**gitee/github**的提交次数应不少于**3次。也就是完成一个阶段版本就应该提交一次，例如1.0版 、2.0版，这样能够在提交时间上体现不同版本的完成时间。

（b） 除了上传代码，还需要有一个项目报告文档，简述项目的设计思路等内容，具体格式参考附件一的样例。

（c） 近期github在国内连接不稳定，虽然国际上通行github，但为了避免风险，建议使用**g**itee或者两者都用（**github可以作为备选**）。**

（d） **建议同学从高级语言程序设计开始，每门专业课程的大作业都提交到自己的**gitee/github账号下。这样今后找工作、升学时，可以在简历中提供相应项目的网址，让读简历的人能够清楚地掌握你的专业能力。同时，这也是你的成长记录。类似一些同学使用在各类ACM竞赛网站上刷题的积分，作为编程能力的证明。

### 2.在B站上的项目讲解视频网址。

完成项目开发后，应录制视频，对项目的问题描述、实际意义、技术路线及算法描述、开发过程中遇到的难点及解决方法、实际进度安排进行讲解，并演示项目实际执行情况，随项目运行演示进一步讲解所开发软件的使用方式及亮点描述等，并将讲解视频上传至B站（https://www.bilibili.com/，不要求实名）。 视频讲解时间**不短于5分钟，不长于8**分钟。视频的标题按照如下规则起名：**【南开大学25C++】XXX，其中**【南开大学2**5C++】**是每位同学都要有的前缀，XXX是同学自主命名的项目名称。 例如，【南开大学25C++】终极2048。

---

# 四、主要流程

## 4.1 游戏内容

### 4.1.1 实体及建筑

修建建筑是游戏发展的基本动作，只能由村民完成。只有市镇中心能生产村民，是游戏的核心建筑。游戏仿造人类文明发展，有人口约束机制，市镇中心可以提供初始的4个人口支持，村民和士兵都计为一个人口。支持更多的人口需要建设房屋，每个房屋可以增加4人口支持。游戏对文明有人口上限限制，原版游戏默认的上限是50，本次设计暂定使用该上限，即村民加军事单位的数量不能超过50。

**建筑图示如下：**


**建筑数据：**

| 建筑名称 | 建造时代 | 建筑大小 | 建造花费 | 建造时间 | 可执行的行动 | 备注 |
|---|---|---|---|---|---|---|
| 市镇中心 | 开局拥有 | 3x3 | - | - | 生产村民、升级时代 | 可以存放资源容纳4人口 |
| 房屋 | 石器时代 | 2x2 | 30木头 | 20s | - | 容纳4人口 |
| 谷仓 | 石器时代 | 3x3 | 120木头 | 30s | 研究箭塔建造 | 用来存放资源 |
| 仓库 | 石器时代 | 3x3 | 120木头 | 30s | 研究攻防技术 | 用来存放资源 |
| 兵营 | 石器时代 | 3x3 | 125木头 | 30s | 训练士兵、升级战斧 | |
| 箭塔 | 工具时代 | 2x2 | 150石头 | 80s | 攻击敌军或建筑 | 需在谷仓解锁科技 |
| 农场 | 工具时代 | 3x3 | 75木头 | 30s | - | 需要先建造市场 |
| 市场 | 工具时代 | 3x3 | 150木头 | 40s | 升级资源采集技术、农场 | 需要先建造谷仓 |
| 靶场 | 工具时代 | 3x3 | 150木头 | 40s | 训练弓手 | 需要先建造兵营 |
| 马厩 | 工具时代 | 3x3 | 150木头 | 40s | 训练骑兵 | 需要先建造兵营 |

### 4.1.2 资源管理和作战系统

游戏内一共有四种资源：粮食、木材、石头、黄金（暂未加入）。游戏中的资源是维持帝国运转的基础，需要合理规划采集和使用，初始拥有200食物，200木头，150石头。资源和单位图示如下：


粮食可以通过击杀动物后对尸体进行采集、采集浆果、农场种地获得，木材通过伐木获得，石头通过开采地图上的石堆获取。只有村民可以采集各种资源。村民在单次采集资源达到上限（基础为10个资源，采集速度为20s/组，部分采集能力可以通过科技升级）后会返回最近的市镇中心、谷仓或仓库存放后，再回到资源处继续采集直到资源采集完成。市镇中心可以**存放各种类型资源**、谷仓可以**存放农场和采集浆果获得的食物**、仓库可以**存放木头、石头以及打猎获得的食物**。因此建议在树林、石堆、猎物旁建造仓库，在浆果丛旁建造谷仓以减少采集资源过程中的行走距离。

兵种目前有棍棒兵、战斧兵、弓箭手、投石兵和侦察骑兵，需要在兵营、靶场、马厩中训练，士兵类型单位只可以进行攻击敌对目标的操作，不能采集资源，且同样占用人口数。每个单位都有生命值、近战/远程攻击力、近战/远程防御（对应类型的防御只能减少受到对应类型攻击时受到的伤害）等属性。己方部队外观为蓝色，敌方部队外观为红色。

**单位数据表：**

| 单位名称 | 血量 | 攻击类型攻击距离 | 攻击力 | 近战防御 | 远程防御 | 建造时间 | 备注 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 瞪羚 | 8 | - | - | - | - | | 死后变为资源 |
| 大象 | 45 | 近战 | 10 | - | - | | 死后变为资源 |
| 村民 | 25 | 远程4/近战 | 4 | - | - | 20s | |
| 棍棒兵 | 40 | 近战 | 3（5） | 0（2） | 0 | 26s | 括号为升级后 |
| 战斧兵 | 50 | 近战 | 5（7） | 0（2） | 0 | 26s | |
| 弓箭手 | 35 | 远程5 | 3 | 0（2） | 0 | 30s | |
| 投石兵 | 25 | 远程4 | 2 | 0 | 2 | 24s | |
| 侦察骑兵 | 60 | 近战 | 3 | 0（2） | 0 | 30s | |
| 短剑手敌方英雄 | 120 | 近战 | 10 | 2 | 2 | | 对建筑攻击力翻倍 |
| 强弓手敌方英雄 | 80 | 远程7 | 6 | 2 | 2 | | 对建筑攻击力翻倍 |
| 骑兵敌方英雄 | 150 | 近战 | 6 | 2 | 2 | | 对建筑攻击力翻倍 |

**资源数据表：**

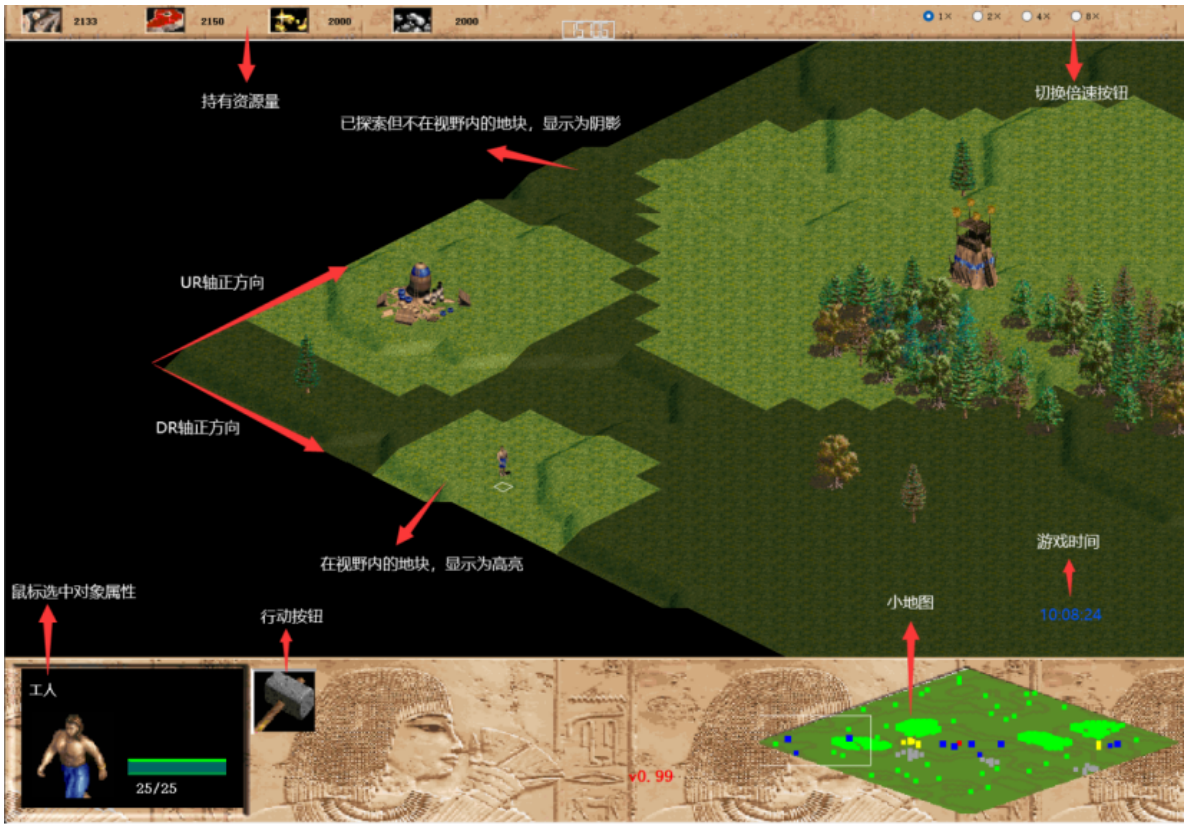
| 资源名称 | 资源类型 | 资源量 | 生成规则 | 备注 |
| --- | --- | --- | --- | --- |
| 树木/森林 | 木头 | 75/棵 | 随机生成零散的树木和两片大树林 | |
| 石堆 | 石头 | 250/块 | 随机生成两大堆石堆 | |
| 浆果丛 | 食物 | 150/丛 | 随机生成一堆、每堆6丛 | |
| 瞪羚尸体 | 食物 | 150/只 | 随机生成一群、每群6只 | 死后才可收集 |
| 大象尸体 | 食物 | 300/只 | 随机生成一群、每群2只 | 死后才可收集 |
| 农场 | 食物 | 250/个 | 玩家自行建造 | 市场科技可以升级农场资源量 |

### 4.1.3 地图和坐标系

游戏以伪三维方式显示二维游戏平面，将正方形平面显示为45度角的菱形。地图大小为72x72格（Block）的草地，采用以最左侧角落为原点，右下方向为DR轴正方向，右上方向为UR轴正方向的平面坐标系。单个整块格子的坐标称为块坐标(BlockDR(int),BlockUR(int))。地形分为斜坡和平地，平地具备0-4的整数高度，斜坡用-1表示。

地图所有格子初始是可见的。玩家持有的单位和建筑会拥有一定距离的视野，处于视野范围内的格子内容将变为可见，当前不在视野内的格子被标记为已探索，在地图上渲染为灰色，其中的对象仍然可见、可被选取。右下角小地图上会显示各种单位的位置（己方为蓝色，树为绿色，动物为黄色，石头、浆果丛为灰色、敌方单位为红色），且敌方单位仅有在视野内才会显示在小地图上。

游戏中的部分对象还拥有细节坐标，以浮点数表示`(DR(double),UR(double))`。每个格子在细节坐标下的尺寸约35.7*35.7(16*根号5,常量表的BLOCKSIDELENGTH)。



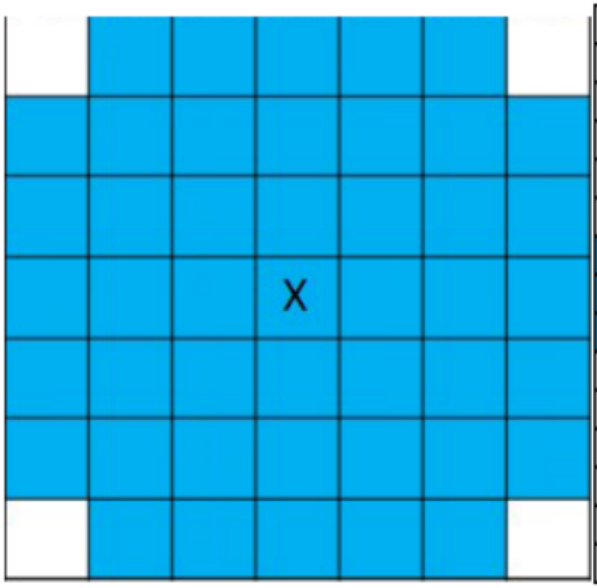

### 4.1.4 视野说明

本游戏采用了按Block简化的街区距离和欧氏距离作为视野系统的基础。在游戏中，每个对象都有一个视野范围，为近似圆形(简化欧氏距离），对小于4的视野做了优化。

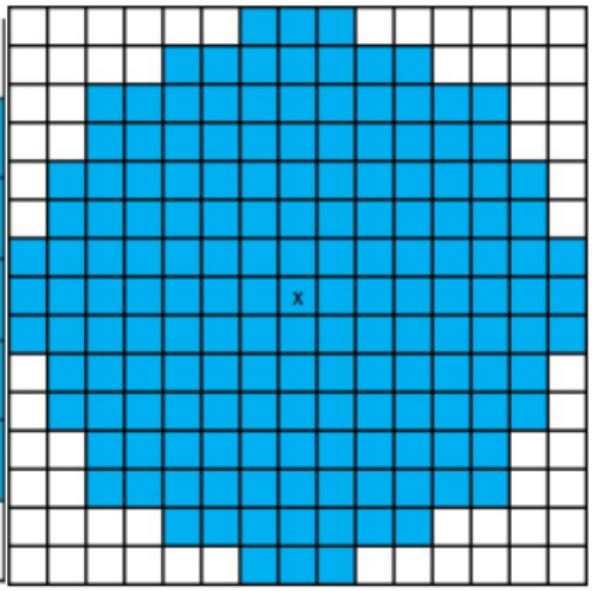
以下是游戏中拥有视野单位的视野表：

| 单位/建筑 | 视野（格） |
| --- | --- |
| 瞪羚 | 2 |
| 市镇中心、谷仓、兵营、市场、房屋、仓库、靶场、农场、马厩、村民、大象、棍棒兵、战斧兵 | 4 |
| 投石兵 | 5 |
| 弓箭手 | 7 |
| 侦察骑兵 | 8 |
| 箭塔 | 10 |

视野为 4 格的图示：




视野为 7 格的图示：




### 4.1.5 游戏目标

玩家在游戏中发送命令控制游戏内的单位、建筑，不断收集资源、建造建筑、研究科技和发展兵力。游戏在第5、10、15分钟会分别派遣一批敌人进攻你的基地，需要控制部队抵御敌对单位的攻击，保护市镇中心。一局游戏总时长不超过25分钟。

**胜利条件**：消灭敌方所有敌人

**失败条件**：己方市镇中心被毁灭或者时间耗尽

**分数计算方法**：胜利按时间计分，时间越短分数越高。

**失败按以下规则记分**：收集某种资源（浆果、羚羊、大象、石头、农田、树木）+5分，资源每收集100个+1分（不足100不计分）。每建设一个住房或农田+1分，其他建筑+2分，升级一个科技+2分。生产一个村民+1分，骑兵+2分，除骑兵外的其他兵种+1分。杀死一个敌人+2分，杀死敌方英雄+10分。游戏运行时分数动态显示。

## 4.2 代码编写

### 阶段1：初始建设与资源采集

- `Farmer.cpp/h` - 农民类，负责基础资源采集

```cpp
#include "Farmer.h"
// 携带资源，5种资源类型，8个方向
std::list<ImageResource>* Farmer::Carry[5][8];
// 行走时，7种状态，8个方向
std::list<ImageResource>* Farmer::Walk[7][8];
// 工作时，7种状态，8个方向
std::list<ImageResource>* Farmer::Work[7][8];
// 站立时，7种状态，8个方向
std::list<ImageResource>* Farmer::Stand[7][8];
// 攻击时，7种状态，8个方向
std::list<ImageResource>* Farmer::Attack[7][8];
// 死亡时，7种状态，8个方向
std::list<ImageResource>* Farmer::Die[7][8];
// 消失时，7种状态，8个方向
std::list<ImageResource>* Farmer::Disappear[7][8];

// 农民类型：村民、伐木工、采集者、矿工、猎人、农民、工人
std::string Farmer::FarmerName[7]={"村民","伐木工","采集者","矿工","猎人","农民","工人"};
// 携带资源类型：空、木材、肉类、石头、黄金
std::string Farmer::FarmerCarry[5]={"","CarryWood","CarryMeat","CarryStone","CarryGold"};

// 点击音效
string Farmer::sound_click = "Click_Villager";

// 工作音效：空、砍伐、采集、采矿、弓箭攻击、耕地、建造
std::string Farmer::sound_work[7] = {\
    "", "Cut", "Gather", "Mine", "Archer_Attack", "Plow", "Build"\
};

// 构造函数：初始化农民对象
Farmer::Farmer(double DR, double UR , Development* playerScience, int playerRepresent )
{
    // 初始化玩家科技和玩家标识
    this->playerScience = playerScience;
    this->playerRepresent = playerRepresent;

    // 设置基础属性
    this->Blood=1;                      // 初始血量
    this->MaxBlood=BLOOD_FARMER;        // 最大血量
    speed = HUMAN_SPEED;                // 移动速度
    this->atk = 3;                      // 攻击力
    attackType = ATTACKTYPE_CLOSE;      // 攻击类型：近战
    phaseFromEnd_MissionAttack = THROWMISSION_FARMER;  // 攻击任务阶段
    vision = VISION_FARMER;             // 视野范围
    ...
}

// 更新帧：处理动画和状态更新
void Farmer::nextframe()
{
    if(isDie())  // 如果已死亡
    {
        if( !isDying() )  // 如果刚开始死亡
        {
            setPreDie();  // 设置预死亡状态
```

```
            requestSound_Die();  // 请求播放死亡音效
        }
            ...
}
// 设置当前图像资源
void Farmer::setNowRes()
{
    std::list<ImageResource> *templist = NULL;

    // 根据当前状态选择对应的图像资源列表
    switch (this->nowstate) {
    case MOVEOBJECT_STATE_STAND:  // 站立状态
        templist=this->Stand[this->state][this->Angle];
        break;
        ...
}

// 获取攻击距离
double Farmer::getDis_attack()
{
    double dis;

    // 如果是远程攻击，基础距离为3
    if(get_AttackType() == ATTACKTYPE_SHOOT) dis = 3;
    else dis = 0;

    ...
}
    ...
```

- `Resource.cpp/h` - 资源基类，管理游戏中的各种资源

```
#include "Resource.h"

// 资源类构造函数
Resource::Resource()
{
}

// 根据资源类型返回应该存放在哪种建筑中
int Resource::get_ReturnBuildingType()
{
    int buildingType;
    switch (resourceSort)
    {
    case HUMAN_WOOD:       // 木材资源
    case HUMAN_GOLD:       // 黄金资源
    case HUMAN_STONE:      // 石头资源
    case HUMAN_STOCKFOOD: // 食物资源
        buildingType = BUILDING_STOCK;  // 存放在仓库中
        break;
    case HUMAN_GRANARYFOOD:  // 谷物资源
        buildingType = BUILDING_GRANARY;  // 存放在粮仓中
        break;
    default:
        buildingType = BUILDING_CENTER;  // 默认存放在城镇中心
        break;
```

```
    }

    return buildingType;
}

```

- `Building_Resource.cpp/h` - 资源建筑类，如伐木场、矿场等

```
#include "Building_Resource.h"
void Building_Resource::nextframe()
{
    setNowRes();
    if(Percent<100)
    {
        nowres = nowlist->begin();
        advance(nowres , (int)(Percent/25));
    }
    ...
}
void Building_Resource::setAttribute()
{
/**
设置产生资源的建筑的各属性：
1 最大血量
2 地基大小
3 产生资源类型
4 当前是否可采集（是）
5 最大资源量
6 当前资源量
*/
    if(Num == BUILDING_FARM)
    {
        MaxBlood = BLOOD_BUILD_FARM;
        Foundation=FOUNDATION_MIDDLE;
        resourceSort = HUMAN_GRANARYFOOD;
        gatherable = false;
        vision = VISION_FARM;
        incorrectNum = false;
        setMaxCnt();
    }
    else incorrectNum = true;
    Cnt = MaxCnt;
}

bool Building_Resource::isGathererAsLandlord(Coordinate* gatherer)
{
    //无主农田，设置地主
    if(this->gatherer == NULL)
    {
        setGatherer(gatherer);
        return true;
    }

    //返回传入采集者是否是地主
    return gatherer == this->gatherer;
}

```

```cpp
void Building_Resource::setMaxCnt()
{
    if(Num == BUILDING_FARM)
    {
        if(playerScience == NULL) MaxCnt = CNT_BUILD_FARM;
        else MaxCnt = CNT_BUILD_FARM + playerScience->get_addition_MaxCnt(getSort(),Num);
    }
}
```

- `StaticRes.cpp/h` - 静态资源类，如树木、金矿等

```cpp
#include "StaticRes.h"
std::list<ImageResource>* StaticRes::staticResource[3];
// 静态资源：浆果丛、石头、金矿
std::string StaticRes::StaticResname[3]={"Bush","Stone","GoldOre"};
std::string StaticRes::StaticResDisplayName[3] = {"浆果丛","石头","金矿"};
// 通过详细坐标创建静态资源
StaticRes::StaticRes(int Num, double DR, double UR)
{
    this->Num=Num;  // 设置资源类型编号

    setDRUR(DR, UR);  // 设置详细坐标
    updateBlockByDetail();  // 根据详细坐标更新块坐标
    setNowRes();  // 设置当前图像资源
    setAttribute();  // 设置资源属性

    // 设置全局编号并注册到全局对象列表
    this->globalNum=g_globalNum;
    g_Object.insert({this->globalNum,this});
    g_globalNum++;
}

// 通过块坐标创建静态资源
StaticRes::StaticRes(int Num, int BlockDR, int BlockUR)
{
    this->Num=Num;
    setBlockDRUR(BlockDR, BlockUR);

    setNowRes();
    ...
}

// 设置静态资源的属性
void StaticRes::setAttribute()
{
    switch(Num){
    case NUM_STATICRES_Bush:  // 浆果丛
        MaxCnt = CNT_BUSH;  // 最大采集量
        resourceSort = HUMAN_GRANARYFOOD;  // 资源类型：食物
        BlockSizeLen = SIZELEN_SINGEL;  // 占地面积：单个单位
        crashLength = CRASHBOX_SINGLEOB;  // 碰撞箱大小：单个单位
        break;
    case NUM_STATICRES_Stone:  // 石头
        MaxCnt = CNT_STONE;  // 最大采集量
        ...
```

```
45 }
```

- `Animal.cpp/h` - 动物类，作为食物来源

```
#include "Animal.h"
// 各种动物状态的图像资源
std::string Animal::Animalname[5]=
{"Tree","Gazelle","Elephant","Lion","Forest"};
std::string Animal::Animalcarcassname[5]=
{"Fallen_Tree","Gazelle","Elephant","Lion","Forest_Stool"};
std::string Animal::AnimalDisplayName[5]={"树","羚羊","大象","狮子","森
林"};
std::list<ImageResource>* Animal::Walk[5][8];       // 行走
std::list<ImageResource>* Animal::Stand[5][8];      // 站立
std::list<ImageResource>* Animal::Attack[5][8];     // 攻击
std::list<ImageResource>* Animal::Die[5][8];        // 死亡
std::list<ImageResource>* Animal::Run[5][8];        // 奔跑
std::list<ImageResource>* Animal::Disappear[5][8];  // 消失
// 音效
std::string Animal::sound_click[5] = {\
    "", "", "Elephant_Stand", "Lion_Stand", ""
};
// 动物属性
// 树木 羚羊， 大象 狮子， 森林
// 最大血量
int Animal::AnimalMaxBlood[5] = { BLOOD_TREE, BLOOD_GAZELLE,
BLOOD_ELEPHANT, BLOOD_LION, BLOOD_FARMER };
// 资源类型
int Animal::AnimalResouceSort[5] = { HUMAN_WOOD, HUMAN_STOCKFOOD,
HUMAN_STOCKFOOD, HUMAN_STOCKFOOD, HUMAN_WOOD };
// 资源数量
int Animal::AnimalCnt[5] = { CNT_TREE, CNT_GAZELLE, CNT_ELEPHANT,
CNT_LION, CNT_FOREST };
// 动画帧间隔
int Animal::AnimalNowresStep[5] = { 0, 0, NOWRES_TIMER_ELEPHANT,
NOWRES_TIMER_LION, 0 };
// 视野范围
int Animal::AnimalVision[5] = { 0, VISION_GAZELLE, VISION_ELEPHANT,
VISION_LION, 0};
   ...
```

## 阶段2：建造房屋与兵营

- `Building.cpp/h` - 建筑基类

```
#include "Building.h"
/**********************静态资源**************************/
// 建筑图像资源列表
std::list<ImageResource>* Building::build[4];  // 建造中的建筑图像
std::list<ImageResource>* Building::built[3][10];  // 已建成的建筑图像，3
个文明，10种建筑
std::list<ImageResource>* Building::buildFire[3];  // 建筑着火时的图像

// 建筑名称数组
```

```
std::string Building::Buildingname[4]={"Small_Foundation","Foundation","Big_Foundation","Building_House1"}; // 地基和房屋名称
std::string Building::Builtname[3][10]={{},  // 已建成建筑名称，3个文明，10种建筑

{"House1","Granary","Center1","Stock","Farm","Market","ArrowTower","ArmyCamp","Stable","Range"},

{"House2","Granary","Center2","Stock","Farm","Market","ArrowTower","ArmyCamp","Stable","Range"}
                                        };
// 建筑显示名称
std::string Building::BuildDisplayName[10]={"房屋","谷仓","市镇中心","仓库","农场","市场","箭塔","兵营","马厩","靶场"};

// 建筑着火效果名称
std::string Building::BuildFireName[3] = { "S_Fire", "M_Fire", "B_Fire"};  // 小、中、大火
   ...
```

- `Human.cpp/h` - 人类单位基类，包括建筑工人

```
#include "Human.h"

// 人类单位默认构造函数
Human::Human()
{
}

// 人类单位构造函数
Human::Human(int Num, double DR, double UR,Development* playerScience, int playerRepresent)
{
    // 初始化玩家科技和玩家标识
    this->playerScience = playerScience;
    this->playerRepresent = playerRepresent;

    // 设置单位基本属性
    this->Num=Num;  // 单位类型编号
    this->DR=DR;    // 下右坐标
    this->UR=UR;    // 上右坐标

    speed = HUMAN_SPEED;  // 设置移动速度
}
// 获取防御力

int Human::getDEF(int attackType_got)
{
    int def = 0;  // 基础防御力

    // 根据攻击类型选择对应的防御属性
    if( attackType_got == ATTACKTYPE_CLOSE||attackType_got == ATTACKTYPE_ANIMAL )
        def = defence_close;  // 近战/动物攻击使用近战防御
    else if(attackType_got == ATTACKTYPE_SHOOT )
        def = defence_shoot;  // 远程攻击使用远程防御
```

```
33
34      // 计算最终防御力：
35      // 1. 基础防御力 * 科技防御加成系数
36      // 2. 加上科技提供的额外防御力
37      return (int)(def * playerScience->get_rate_Defence(getSort(),Num,ARMY_INFANTRY , attackType_got) ) +\
38              playerScience->get_addition_Defence(getSort() , Num , ARMY_INFANTRY , attackType_got);
39  }
40
```

- `Player.cpp/h` - 玩家类，管理建筑权限和资源

```
1   #include "Player.h"
2   // 构造函数
3   Player::Player()
4   {
5       playerScience = new Development(represent);  // 创建玩家科技系统
6   }
7
8   // 玩家构造函数
9   Player::Player(int represent)
10  {
11      this->represent = represent;  // 设置玩家标识
12      playerScience = new Development(represent);  // 创建玩家科技系统
13  }
14
15  // 玩家析构函数
16  Player::~Player()
17  {
18      delete playerScience;  // 释放科技系统内存
19
20      // 清空所有对象池
21      list<Human*>::iterator iter_deleHuman = human.begin();
22      list<Building*>::iterator iter_deleBuild = build.begin();
23      list<Missile*>::iterator iter_deleMissile = missile.begin();
24
25      // 删除所有人类单位
26      while(iter_deleHuman!=human.end())iter_deleHuman = deleteHuman(iter_deleHuman);
27      // 删除所有建筑
28      while(iter_deleBuild!=build.end())iter_deleBuild = deleteBuilding(iter_deleBuild);
29      // 删除所有投射物
30      while(iter_deleMissile!=missile.end())iter_deleMissile = deleteMissile(iter_deleMissile);
31  }
32  //添加实例对象
33  // 添加建筑
34
35  Building* Player::addBuilding(int Num, int BlockDR, int BlockUR , double percent)
36  {
37      Building *newbuilding = NULL;
38      // 根据建筑类型创建不同的建筑对象
39      if(Num == BUILDING_FARM)
```

```
        newbuilding = new Building_Resource(Num,BlockDR,BlockUR,getCiv() , playerScience , represent , percent);  // 农场
    else
        newbuilding=new Building(Num,BlockDR,BlockUR, getCiv(), playerScience , represent , percent);  // 其他建筑

    build.push_back(newbuilding);  // 添加到建筑列表
    return newbuilding;
}

// 添加人类单位
int Player::addHuman(int Num, double DR, double UR)
{
    Human *newhuman=new Human(Num,DR,UR , playerScience , represent);  // 创建人类单位
    ...
}
    ...
```

- `Development.cpp/h` - 发展类，管理科技树和建筑解锁

```
#include "Development.h"

// 科技系统默认构造函数
Development::Development()
{
    init_DevelopLab();  // 初始化科技树
}

// 科技系统构造函数
Development::Development(int represent)
{
    playerRepresent = represent;  // 设置玩家标识
    init_DevelopLab();  // 初始化科技树
}

// 获取移动速度倍率
double Development::get_rate_Move(int sort , int type)
{
    double rate = 1;  // 基础移动速度倍率
    if(sort == SORT_FARMER) rate += rate_FarmerMove;  // 农民获得额外移动速度加成

    return rate;
}

// 获取生命值倍率
// 参数说明:
// sort: 单位类型
// type: 具体单位编号
double Development::get_rate_Blood(int sort , int type)
{
    double rate = 1;  // 基础生命值倍率
    if(sort == SORT_FARMER) rate+= rate_FarmerBlood;  // 农民获得额外生命值加成

```

```
    return rate;
}

// 获取生命值加成
int Development::get_addition_Blood( int sort , int type )
{
    int addition = 0;  // 基础生命值加成
    return addition;
}

// 获取攻击力倍率
double Development::get_rate_Attack( int sort , int type , int
armyClass , int attackType, int interSort, int interNum )
{
    double rate = 1;  // 基础攻击力倍率

    // 对建筑的特殊攻击加成
    if(interSort!=-1 && interNum!=-1)
    {
        if(interSort == SORT_BUILDING && sort == SORT_ARMY && (type ==
AT_SWORDSMAN || type == AT_CAVALRY || type == AT_IMPROVED))
            rate = 2;  // 剑士、骑兵和改良单位对建筑造成双倍伤害
    }

    return rate;
}
   ...
```

### 阶段3：防御与士兵训练 （因篇幅原因剩余代码不再给出，详细代码可点击 github.com

- `Army.cpp/h` - 军队单位类，包括各种士兵
- `Bloodhaver.cpp/h` - 拥有血量的对象基类，用于战斗单位
- `Missile.cpp/h` - 投射物类，用于远程攻击
- `Building.cpp/h` - 建筑基类中的防御建筑部分

### 阶段4：农田与市场建设

- `Building_Resource.cpp/h` - 资源建筑类中的农田和市场
- `Farmer.cpp/h` - 农民类中的农田工作
- `Resource.cpp/h` - 资源类中的食物和贸易资源
- `Development.cpp/h` - 发展类中的农业和市场科技

### 阶段5：总攻与敌方英雄击杀

- `Army.cpp/h` - 军队单位类，包括英雄单位
- `EnemyAI.cpp/h` - 敌人AI，管理敌方单位行为
- `UsrAI.cpp/h` - 用户AI，管理己方单位行为
- `Missile.cpp/h` - 投射物类，用于攻击敌方单位
- `Core.cpp/h` - 核心游戏逻辑，处理战斗和击杀判定
- `Core_CondiFunc.cpp/h` - 核心条件函数，处理战斗相关条

### 还有一些基础支持的类，贯穿整个游戏过程：

- `Coordinate.cpp/h` - 坐标系统
- `Map.cpp/h` - 地图管理

- `Block.cpp/h` - 地图块
- `MoveObject.cpp/h` - 可移动对象基类
- `GlobalVariate.cpp/h` - 全局变量
- `config.h` - 配置文件
- `digitalConfig.h` - 数字化配置

## 4.3 操作说明

### 4.3.1鼠标控制

左键：选中单位（村民、动物等）、资源块（树、石头等）、建筑以查看对象的属性，或在已选中特定对象时执行快捷栏的行动命令（例如：选中市镇中心后执行“创造村民”操作）。

右键：移动或命令。点击鼠标右键，可以命令选中的可移动单位（村民、士兵）前往某一位置，或者命令其采集/攻击点击的对象。

### 4.3.2Debug调试信息

本系统包含了一个可以显示彩色文本的调试信息框，位于界面的右侧。调试信息的构成分为两部分：当前游戏已运行时间+具体信息，按信息种类以不同的颜色显示。

**游戏基本事件:(显示为蓝色)**

```
游戏开始:游戏开始
技术和升级完成:已完成技术升级：技术名
产生农民完成:产生了新的农民 FARMER_XXXXX(SN编号)
建筑建造完:BUILDING_XXX 1xxxx 建造完成
游戏目标达成/未达成:游戏失败/游戏胜利
```

**AI处理部分:(显示为绿色)**

```
BuildingAction:<下达指令：>BUILDING_XXX XXXXX 执行行动 ACTION
HumanMove:<下达指令：>FARMER XXXXX 移动至 （DR，UR）
HumanAction:<下达指令：>FARMER XXXXX 设置工作目标为 类型 XXXXX
HumanBuild:<下达指令：>FARMER XXXXX 开始建造 BUILDING_ XXXXX
```

**意外信息:(显示为红色)**

```
碰撞停止:类型 XXXXX 与 类型 XXXXX 碰撞而停止
攻击:FARMER XXXXX 被 ANIMAL_ XXXXX 攻击
动物死亡:ANIMAL_ XXXXX 死亡
农民死亡:FARMER_ XXXXX 死亡
资源采完:类型（COORES_BUSH/ANIMAL_GAZELLE）XXXXX 采集完成
建造失败:失败理由
```

**其他按钮:**

游戏界面右上角可以切换游戏倍速(1、2、4、8）以及暂停游戏，暂停时游戏会绘制格子边界以便查看。

游戏界面下方的“关于”按钮可以查看作者信息，“设置”按钮可以导出调试栏信息文件。

# 五、单元测试

1. **资源采集模块**:

- 测试村民能否正确采集浆果、木材，并自动往返存储资源。
- 验证资源计数（`myInfo.Wood`、`myInfo.Meat`）是否实时更新。

2. **建筑建造模块**：
   - 检查建筑位置是否合法（非斜坡、无碰撞）。
   - 验证`HumanBuild`返回值是否为正值（成功）或错误码（如`ACTION_INVALID_OVERLAP`）。
3. **士兵训练模块**：
   - 测试兵营能否正常生产棍棒兵，升级后能否训练斧头兵。
   - 检查人口上限是否被正确计算（`Human_MaxNum`）。
4. **防御系统测试**：
   - 验证箭塔能否自动攻击进入视野的敌人。
   - 测试士兵在`HUMAN_STATE_STOP`状态下是否重新分配任务。

**错误处理代码示例：**

```
// 检查指令执行结果
if (id_Wrong_lastFrame >= 0) {
    int ret = myInfo.ins_ret[id_Wrong_lastFrame];
    if (ret == ACTION_INVALID_RESOURCE) {
        DebugText("资源不足，无法执行命令！");
    }
}
```

---

# 六、收获

1. **技术能力提升**：
   - 掌握了Qt框架的多线程编程（主线程与AI线程并行）。
   - 学会熟练使用STL容器管理动态数据（如`vector<tagBuilding>`）。
   - 实践即时战略游戏AI设计，包括状态机控制、资源调度、路径规划等知识。
2. **策略优化经验**：
   - 学习了如何高效布局建筑（如农田围绕市镇中心）来减少村民移动时间。
   - 实现了近战单位优先攻击敌方远程单位的士兵分组调度。
3. **调试与性能优化**：
   - 学会使用`DebugText`输出关键信息，快速定位逻辑错误。
   - 在Release模式下优化代码性能，确保8倍速运行时指令响应及时。

---

**总结**：这是主播第一次使用C++编程和Qt框架编写的小游戏，灵感来源于小时候玩过的红警游戏。这个项目在很早之前就已经开始做了，期间也进行了很多修改和调整。尽管游戏目前还存在许多不足，但已经是我在有限时间内能做到的最好的完成度了。相比使用java和python编写游戏，c++代码更加繁琐，很多工具也需要自己编写。后续在游戏制作中我应该不会优先考虑c++语言了。感谢很多朋友和老师的帮助，愿我们都有美好的未来。