# 2025年C++课程设计单人任务

NewAOE 2.50版说明书

## 1. 简介

本设计单人任务使用即时战略游戏搭建的博弈系统，测试学生的C++编程水平。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2.5版本复刻了原游戏石器时代和工具时代的绝大部分内容和操作，并开发了通过程序控制游戏过程的接口（以下简称为AI接口）。

本任务的目标是让同学们通过AI接口编写一系列代码控制游戏过程，并完成指定的游戏目标。AI程序在博弈系统中运行的成绩，将作为C++课程设计个人项目评分的基础。

## 2. 游戏内容与操作

### 2.1实体及建筑



图1 建筑图示

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 船坞 | 石器时代 | 2x2 | 120木头 | 40s | 生产水上单位 | 建在海上，靠岸边 |

表1 建筑数据

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、船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。



图2.1 我方大陆资源和单位图示



图2.2 敌方大陆沿岸资源和单位图



图2.3 敌方大陆内陆资源和单位图示

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | 木船和运输船也属于村民 |
| 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 |  |
| 战船 | 300 | 远程10 | 20 | - | - | 30s |  |

表2 单位数据

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |  |
| 渔场 | 食物 | 200/个 | 少量渔场生成在岸边，村民可捕捞。其余需通过渔船捕捞 |  |
| 石堆 | 石头 | 250/块 | 随机生成两大堆石堆 |  |
| 金矿 | 黄金 | 200/块 | 生成在海洋对岸的岛屿上，有重兵守护 | 游戏目标 |
| 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 |

表3 资源数据

### 2.2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（游戏目标）。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，0黄金。

粮食可以通过击杀动物后对尸体进行采集、在渔场捕鱼、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，金子通过开采远方岛屿上的金矿获取。渔场可以由渔船和村民采集，其他各种资源只有村民可以采集。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）后会返回最近的相关建筑存放后，再回到资源处继续采集直到资源采集完成。市镇中心可以存放所有类型资源、谷仓可以存放农场获得的食物、仓库可以存放木头、石头、金子以及打猎、打鱼获得的食物、船坞可以存放捕鱼获得的食物。因此建议在树林、石堆、金矿、猎物旁建造仓库，在渔场旁建造船坞以减少采集资源过程中的行走距离。

游戏兵种有棍棒兵、战斧兵、弓箭手、投石兵、骑兵和战船，需要在兵营、靶场、马厩、船坞中生产或训练，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。

### 2.3时代特性

在石器时代，玩家的建造与科技升级受到以下规则限制：

1. 可建造的建筑物仅包括：仓库、谷仓、兵营、船坞和房屋。

2. 兵营在该时代只能训练棍棒兵，无法进行斧头兵的科技升级。

3. 船坞在该时代只能建造渔船，无法生产其他单位。

4. 仓库与谷仓均不具备任何科技升级动作，仅提供基本功能。

玩家必须至少建造船坞、仓库、兵营和谷仓中的任意两座建筑，并消耗500单位粮食，方可进行时代升级。

进入工具时代后，玩家可以建造新建筑市场、农田、靶场、马厩、箭塔，可以在已有建筑中进行兵种和技术升级，生产新的单位。

### 2.4地图和坐标系

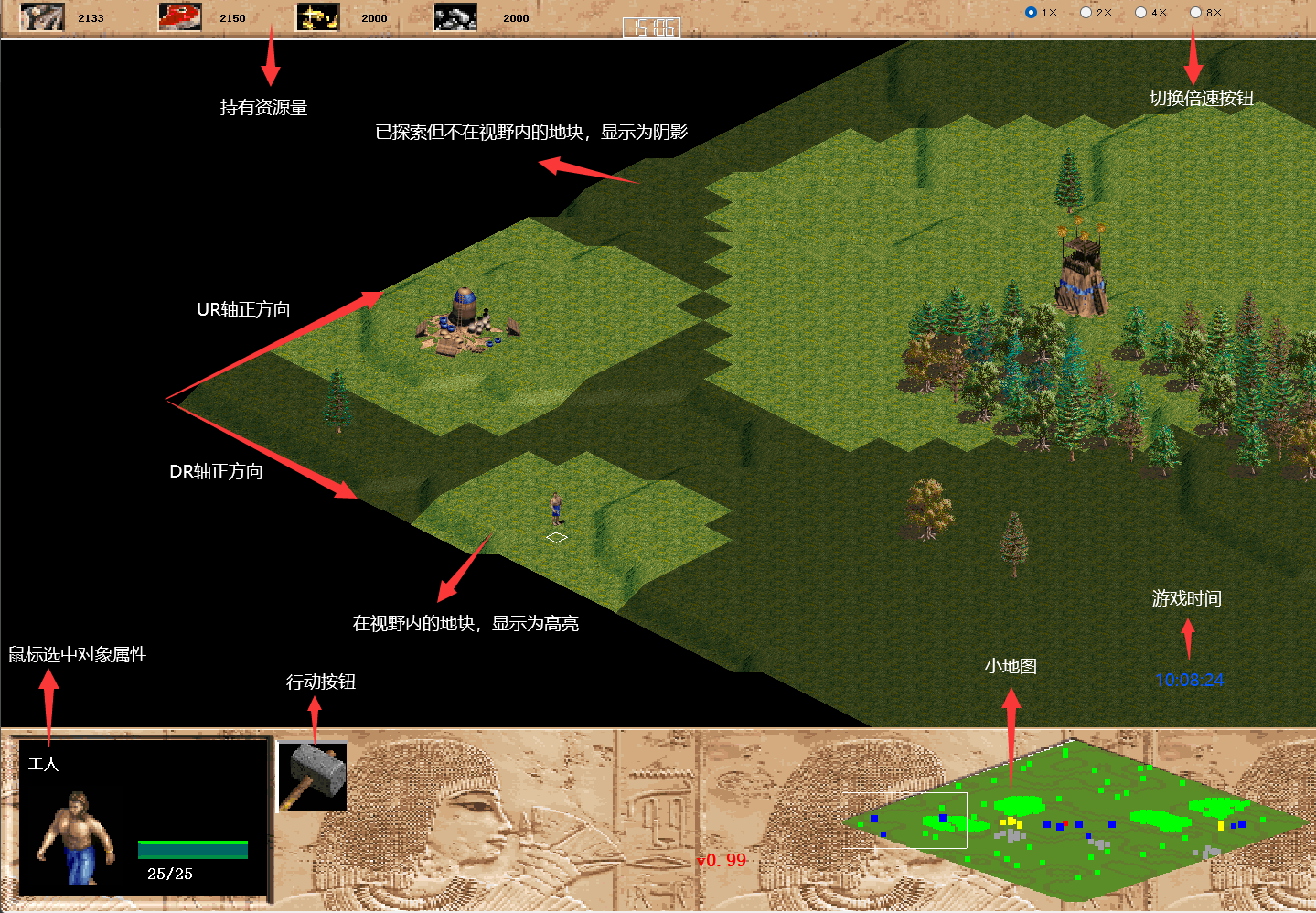


图3 坐标系和系统界面

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128x128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))。地形分为海洋和陆地，陆地具备0-4的整数高度，斜坡用-1表示。

地图上，市政中心所在大陆的格子初始是已探索的。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，石头为灰色、黄金为金色，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。海外范围是未知的，标记为黑色，我方单位或者建筑会依照自己的视野将未知部分标记为已探索。

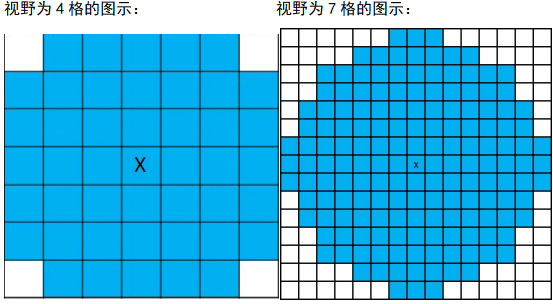
游戏中的部分对象还拥有细节坐标，以浮点数表示(DR(double),UR(double))。每个格子在细节坐标下的尺寸约35.7\*35.7(16\*根号5,常量表的BLOCKSIDELENGTH)。

### 2.5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、船坞 | 4 |
| 投石兵 | 5 |
| 弓箭手 | 7 |
| 侦察骑兵 | 8 |
| 箭塔 | 10 |



### 2.6鼠标控制

鼠标控制仅为手动运行游戏时操控使用，使用AI代码操控时需按照AI接口的代码规范操作发送命令。

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。

### 2.7 Debug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

玩家自定义信息：

可以在AI代码内使用相关接口在Debug栏输出自定义提示信息，便于调试。

### 2.8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。

## 3. 游戏目标与评分准则

学生编写AI逻辑代码即时获取游戏状态，并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。一局游戏总时长不超过25分钟。

胜利条件：突破敌方对金矿的防守，收集足够多的黄金

失败条件：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（鱼、羚羊、大象、石头、黄金、农田作物、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。生产一个村民或渔船+1分，骑兵、战船、运输船+2分，其他兵种+1分。杀死一个敌人+2分。登陆敌方大陆+10分，采集到黄金额外+10分。游戏运行时分数动态显示。最终成绩由各班老师制定规则转换为百分制，每个班内设定胜利分数线，确保胜利者得分大于失败者。

## 4. AI接口

本博弈系统采取了多线程并行的方式完成代码游戏控制。主线程为博弈系统本体，负责游戏的绘制、规则演绎、手动操作，以及响应用户线程的命令。次线程为用户编程线程，设计了一个类名为UsrAI的接口类，请在成员函数UsrAI::processData()中连续获取游戏信息，再根据自己拟定的策略调用命令函数，控制游戏过程。

### 4.1获取游戏信息的接口

函数UsrAI::tagInfo getInfo()提供游戏当前帧的数据。

用户需要定义一个tagInfo类型的变量，获得此函数的返回值，如：

tagInfo info = getInfo();

建议在建议在processData()的第一句话获得当前帧的游戏状态。

接口返回结构体结构说明：

tagInfo是总结构，包含tagBuilding、tagResource、tagFarmer、tagArmy等结构体，各个结构代码如下。其中对于Type、ProductSort等分类参数可通过查询常量表（在本部分最后）来获得对应关系。

struct tagInfo

{

vector<tagBuilding> buildings; // 我方建筑列表

vector<tagFarmer> farmers; // 我方农民列表，包括运输船和渔船

vector<tagArmy> armies; // 我方军队列表，包括战船

vector<tagBuilding> enemy\_buildings; // 敌方建筑列表

vector<tagFarmer> enemy\_farmers; // 敌方农民列表

vector<tagArmy> enemy\_armies; // 敌方军队列表

vector<tagResource> resources; // 资源列表

//以上列表中的元素顺序每一帧都是随机的

map<int, int> ins\_ret; // 指令返回值，map<id, ret>

TerrainData\* theMap; // 一个指向二维数组的指针,内部元素为tagTerrain类型，定义见后文，表示地图信息，[BlockDR][BlockUR]。

int GameFrame; // 当前帧数

int civilizationStage; // 文明阶段

// #define CIVILIZATION\_STONEAGE 1

// #define CIVILIZATION\_TOOLAGE 2

int Wood; // 木材数量

int Meat; // 肉类数量

int Stone; // 石头数量

int Gold; // 黄金数量

int Human\_MaxNum; // 最大人口数量

};

struct tagTerrain {

    int height; // 如过type是MAPPATTERN\_GRASS，则为0，1，2，3（表示高度），否则为-1

int type; // 宏：MAPPATTERN\_UNKNOWN（未知区域，地图渲染为黑色）、MAPPATTERN\_OCEAN（海洋区域）、MAPPATTERN\_GRASS（陆地区域）

};

struct tagBuilding

{

int BlockDR,BlockUR; //区块坐标

int Type; // 建筑类型

int SN; // 序列号

int Blood; // 当前血量

int MaxBlood; // 最大血量

int Percent; // 完成百分比

int Project; // 当前项目

int ProjectPercent; // 项目完成百分比

int Cnt; // 剩余资源量（仅农田）

};

struct tagResource

{

double DR,UR; //细节坐标

int BlockDR,BlockUR; //区块坐标

int Type; // 资源类型

int SN; // 序列号

int ProductSort; // 产品种类

int Cnt; // 剩余资源数量

int Blood; // 当前血量

};

struct tagHuman

{

double DR,UR; //细节坐标

int BlockDR,BlockUR; //区块坐标

double DR0,UR0; // 目的地坐标

int NowState; // 当前状态

int WorkObjectSN; // 工作对象序列号

int Blood; // 当前血量

int SN; // 序列号

int attack; // 攻击力

int rangedDefense; // 远程防御

int meleeDefense; // 近战防御

};

struct tagFarmer: public tagHuman

{

int ResourceSort; // 手持资源种类

int Resource; // 手持资源数量；如果是运输船，则代表单位数量

int FarmerSort; // 村民的类型： 0表示陆地生产单位，1表示运输船，2表示渔船

};

struct tagArmy:public tagHuman

{

int Sort; //军队种类，详细定义见后文

};

### 4.2发送命令函数及其参数和返回值

UsrAI类提供了4个Action成员函数向主线程发出控制命令。

1. 对建筑SN发出命令：（Action数值见“建筑动作类型常量表”）

int BuildingAction(int SN, int Action)

1. 命令单位SN走向指定细节坐标（DR0, UR0）：

int HumanMove(int SN, double DR0, double UR0)

对于陆地单位，合法目的地为可达的已探索陆地。。

对于海洋单位，合法目的地为可达的已探索海洋和可达的已探索海岸（沿海的陆地）。

若目的地非法，则为未定义行为(Undefined Behavior)，不保证能到达目的地。

1. 将编号为obSN的对象设定为单位SN的对象：

int HumanAction(int SN, int obSN)

村民会自动走向对象并工作，士兵会自动走向对象进攻

若SN与obSN相同，则该单位自我销毁

SN为单位，obSN为运输船时，该指令为命令SN登上obSN

此命令也可以设置箭塔的攻击目标为obSN

1. 命令村民SN在块坐标BlockL,BlockU处建造类型为BuildingNum的新建筑：

int HumanBuild(int SN, int BuildingNum, int BlockDR, int BlockUR)

这4个函数的第一个参数都是SN，为执行命令动作的对象编号（所有对象的唯一编号，通过提供获取get函数所返回的结构体中的元素获取），并返回指令的编号（详见后文）。

系统还提供了几个调试函数：

void DebugText(std::string debugstr)

void DebugText (int debugInt)

void DebugText (double debugDouble)

用于在侧边栏显示用户信息。

为方便方便调试，系统设计了以下作弊函数用于加快游戏进度

void cheatRes();  //调用此函数后，各类资源+500

void cheatAction();  //调用此函数后，之后下达的所有指令都不消耗时间

正式测试时作弊函数关闭失效。

### 4.3编程所需的常量表

1、建筑类型BuildingNum**常量表：**

#define BUILDING\_HOME 0 //房子

#define BUILDING\_GRANARY 1 //谷仓

#define BUILDING\_CENTER 2 //市镇中心

#define BUILDING\_STOCK 3 //仓库

#define BUILDING\_FARM 4 //农田

#define BUILDING\_MARKET 5 //市场

#define BUILDING\_ARROWTOWER 6 //箭塔

#define BUILDING\_ARMYCAMP 7 //兵营

#define BUILDING\_STABLE 8 //马厩

#define BUILDING\_RANGE 9 //靶场

#define BUILDING\_DOCK 10 //船坞

2、BuildingAction函数的**Action**常量：

#define BUILDING\_CENTER\_CREATEFARMER 1 //市镇中心生产村民

#define BUILDING\_CENTER\_UPGRADE 2 //市镇中心升级时代

#define BUILDING\_GRANARY\_ARROWTOWER 3 //谷仓研发箭塔

#define BUILDING\_MARKET\_WOOD\_UPGRADE 4 //市场升级伐木（加远程攻击距离）

#define BUILDING\_MARKET\_STONE\_UPGRADE 5//市场升级采石（增加投石兵距离和伤害）

#define BUILDING\_MARKET\_FARM\_UPGRADE 6//市场升级农田（增加农田容量）

#define BUILDING\_STOCK\_UPGRADE\_USETOOL 8//仓库升级使用工具（增加近战伤害）

#define BUILDING\_STOCK\_UPGRADE\_DEFENSE\_INFANTRY 9//仓库升级步兵护甲

#define BUILDING\_STOCK\_UPGRADE\_DEFENSE\_ARCHER 10//仓库升级弓兵护甲

#define BUILDING\_STOCK\_UPGRADE\_DEFENSE\_RIDER 11//仓库升级骑兵护甲

#define BUILDING\_ARMYCAMP\_CREATE\_CLUBMAN 13//军营生产棍棒兵/斧头兵

#define BUILDING\_ARMYCAMP\_CREATE\_SLINGER 14//军营生产投石兵

#define BUILDING\_ARMYCAMP\_UPGRADE\_CLUBMAN 15//军营升级棍棒兵为斧头兵

#define BUILDING\_RANGE\_CREATE\_BOWMAN 16 //靶场生产弓箭手

#define BUILDING\_STABLE\_CREATE\_SCOUT 17 //马厩生产侦察骑兵

#define BUILDING\_DOCK\_CREATE\_SAILING 119 //船坞建造渔船

#define BUILDING\_DOCK\_CREATE\_WOOD\_BOAT 120 //船坞建造运输船

#define BUILDING\_DOCK\_CREATE\_SHIP 515 //船坞建造战船

3、建筑信息及建筑行动所需资源，可以在调用行动命令之前检查资源是否足够

/\*\*市政中心\*\*/

#define BLOOD\_BUILD\_CENTER 600 //血量

#define VISION\_CENTER 4 //视野（block）

#define BUILD\_CENTER\_WOOD 200 //建造所需木头

#define TIME\_BUILD\_CENTER 60 //建造时间（单位秒）

//生产村民

#define BUILDING\_CENTER\_CREATEFARMER\_FOOD 50 //行动所需食物

#define TIME\_BUILDING\_CENTER\_CREATEFARMER 20 //行动所需时间

以下常量无注释的，请参看上面常量的注释，结合常量名理解

/\*\*房子\*\*/

#define BLOOD\_BUILD\_HOUSE 75

#define VISION\_HOME 3

#define BUILD\_HOUSE\_WOOD 30

#define TIME\_BUILD\_HOME 20

/\*\*仓库\*\*/

#define BLOOD\_BUILD\_STOCK 350

#define VISION\_STOCK 3

#define BUILD\_STOCK\_WOOD 120

#define TIME\_BUILD\_STOCK 30

//升级工具利用（1级）

#define BUILDING\_STOCK\_UPGRADE\_USETOOL\_FOOD 100

#define TIME\_BUILDING\_STOCK\_UPGRADE\_USETOOL 40

#define BUILDING\_STOCK\_UPGRADE\_USETOOL\_ADDITION\_ATKCLOSE 2

//升级步兵护甲（1级）

#define BUILDING\_STOCK\_UPGRADE\_DEFENSE\_INFANTRY\_FOOD 75

#define TIME\_BUILDING\_STOCK\_UPGRADE\_DEFENSE\_INFANTRY 40

#define BUILDING\_STOCK\_UPGRADE\_DEFENSE\_INFANTRY\_ADDITION\_DEFENSE\_INFANTRY 2

//升级弓兵护甲（1级）

#define BUILDING\_STOCK\_UPGRADE\_DEFENSE\_ARCHER\_FOOD 100

#define TIME\_BUILDING\_STOCK\_UPGRADE\_DEFENSE\_ARCHER 40

#define BUILDING\_STOCK\_UPGRADE\_DEFENSE\_ARCHER\_ADDITION\_DEFENSE\_ARCHER 2

//升级骑兵护甲（1级）

#define BUILDING\_STOCK\_UPGRADE\_DEFENSE\_RIDER\_FOOD 125

#define TIME\_BUILDING\_STOCK\_UPGRADE\_DEFENSE\_RIDER 30

#define BUILDING\_STOCK\_UPGRADE\_DEFENSE\_RIDER\_ADDITION\_DEFENSE\_RIDER 2

/\*\*谷仓\*\*/

#define BLOOD\_BUILD\_GRANARY 350

#define VISION\_GRANARY 4

#define BUILD\_GRANARY\_WOOD 120

#define TIME\_BUILD\_GRANARY 30

//研发箭塔

#define BUILDING\_GRANARY\_ARROWTOWER\_FOOD 50

#define TIME\_BUILDING\_GRANARY\_RESEARCH\_ARROWTOWER 10

/\*\*兵营\*\*/

#define BLOOD\_BUILD\_ARMYCAMP 350

#define VISION\_ARMYCAMP 4

#define BUILD\_ARMYCAMP\_WOOD 125

#define TIME\_BUILD\_ARMYCAMP 30

//生产棍棒兵

#define BUILDING\_ARMYCAMP\_CREATE\_CLUBMAN\_FOOD 50

#define TIME\_BUILDING\_ARMYCAMP\_CREATE\_CLUBMAN 26

//升级棍棒兵为斧头兵

#define BUILDING\_ARMYCAMP\_UPGRADE\_CLUBMAN\_FOOD 100

#define TIME\_BUILDING\_ARMYCAMP\_UPGRADE\_CLUBMAN 40

//生产投石者

#define BUILDING\_ARMYCAMP\_CREATE\_SLINGER\_FOOD 40

#define BUILDING\_ARMYCAMP\_CREATE\_SLINGER\_STONE 10

#define TIME\_BUILDING\_ARMYCAMP\_CREATE\_SLINGER 24

/\*\*靶场\*\*/

#define BLOOD\_BUILD\_RANGE 350

#define VISION\_RANGE 3

#define BUILD\_RANGE\_WOOD 150

#define TIME\_BUILD\_RANGE 40

//生产弓箭手

#define BUILDING\_RANGE\_CREATE\_BOWMAN\_FOOD 40

#define BUILDING\_RANGE\_CREATE\_BOWMAN\_WOOD 20

#define TIME\_BUILDING\_RANGE\_CREATE\_BOWMAN 30

/\*\*马厩\*\*/

#define BLOOD\_BUILD\_STABLE 350

#define VISION\_STABLE 3

#define BUILD\_STABLE\_WOOD 150

#define TIME\_BUILD\_STABLE 40

//生产侦察骑兵

#define BUILDING\_STABLE\_CREATE\_SCOUT\_FOOD 100

#define TIME\_BUILDING\_STABLE\_CREATE\_SCOUT 30

/\*\*市场\*\*/

#define BLOOD\_BUILD\_MARKET 350

#define VISION\_MARKET 4

#define BUILD\_MARKET\_WOOD 150

#define TIME\_BUILD\_MARKET 40

//升级伐木（1级）

#define BUILDING\_MARKET\_WOOD\_UPGRADE\_FOOD 120

#define BUILDING\_MARKET\_WOOD\_UPGRADE\_WOOD 75

#define TIME\_BUILDING\_MARKET\_UPGRADE\_CUTTING 40

#define BUILDING\_MARKET\_WOOD\_UPGRADE\_ADDITION\_CARRY 2

#define BUILDING\_MARKET\_WOOD\_UPGRADE\_ADDITION\_DISSHOOT 1

//升级采集石头（1级）

#define BUILDING\_MARKET\_STONE\_UPGRADE\_FOOD 100

#define BUILDING\_MARKET\_STONE\_UPGRADE\_STONE 50

#define TIME\_BUILDING\_MARKET\_UPGRADE\_DIGGINGSOTNE 60

#define BUILDING\_MARKET\_STONE\_UPGRADE\_ADDITION\_CARRY 3

#define BUILDING\_MARKET\_STONE\_UPGRADE\_ADDITION\_SILNGERATK 1

#define BUILDING\_MARKET\_STONE\_UPGRADE\_ADDITION\_SILNGERDIS 1

//升级农田（1级）

#define BUILDING\_MARKET\_FARM\_UPGRADE\_FOOD 200

#define BUILDING\_MARKET\_FARM\_UPGRADE\_WOOD 50

#define TIME\_BUILDING\_MARKET\_UPGRADE\_FARM 60

#define BUILDING\_MARKET\_FARM\_UPGRADE\_ADDITION\_FOOD 75

/\*\*农场\*\*/

#define BLOOD\_BUILD\_FARM 50

#define CNT\_BUILD\_FARM 250 //农田产生的基础资源量

#define VISION\_FARM 3

#define BUILD\_FARM\_WOOD 75

#define TIME\_BUILD\_FARM 30

/\*\*箭塔\*\*/

#define BLOOD\_BUILD\_ARROWTOWER 125

#define VISION\_ARROWTOWER 10

#define DIS\_BUILD\_ARROWTOWER 5 //对近战防御力

#define ATK\_BUILD\_ARROWTOWER 3 //攻击力

#define DEFSHOOT\_BUILD\_ARROWTOWER 3 //对箭矢类远程防御力

#define BUILD\_ARROWTOWER\_STONE 150

#define TIME\_BUILD\_ARROWTOWER 80

/\*\*船坞\*\*/

#define BLOOD\_BUILD\_DOCK 350

#define VISION\_DOCK 4

#define BUILD\_DOCK\_WOOD 120

#define TIME\_BUILD\_DOCK 40

//生产战船

#define BUILDING\_DOCK\_CREATE\_SHIP\_WOOD 60

#define TIME\_BUILDING\_DOCK\_CREATE\_SHIP 30

//生产渔船

#define BUILDING\_DOCK\_CREATE\_SAILING\_WOOD 60

#define TIME\_BUILDING\_DOCK\_CREATE\_SAILING 30

//生产运输船

#define BUILDING\_DOCK\_CREATE\_WOOD\_BOAT\_WOOD 60

#define TIME\_BUILDING\_DOCK\_CREATE\_WOOD\_BOAT 30

1. 军队种类（tagArmy::sort 存储的信息）

#define AT\_CLUBMAN 0 //棍棒兵

#define AT\_SLINGER 1 //投石兵

#define AT\_SWORDSMAN 4 //战斧兵

#define AT\_BOWMAN 2 //弓兵

#define AT\_SCOUT 3 //侦察骑兵

#define AT\_SHIP 7 //战船

1. AI控制函数错误码

Action成员函数会返回一个大于0的ID，作为该命令的唯一标识，供用户在下一帧检查该命令是否执行成功。在发出命令的下一帧，即再次进入processData()时，可获取Info直接用Info.ins\_ret[id]查询，得到该命令的执行结果数值获取详细的错误码信息。错误码仅适合在调试时检查，实际编写代码时应该避免依赖错误码来处理人物逻辑。错误码常量表如下：

#define ACTION\_INVALID\_SN -1 // SN不存在或非法控制SN

#define ACTION\_INVALID\_ACTION -2 //Action不存在

#define ACTION\_INVALID\_LOCATION -3 //指定位置超界

#define ACTION\_INVALID\_OBSN -4 //obSN不存在或obSN非可执行行动对象

#define ACTION\_INVALID\_BUILDINGNUM -5 //BuildingNum不存在

#define ACTION\_INVALID\_RESOURCE -6 //资源不足

#define ACTION\_INVALID\_NULLWORKER -80 //控制对象被删除

#define ACTION\_INVALID\_NULLGOALOBJECT -81 //目标对象已被删除

#define ACTION\_INVALID\_ISNTFREE -82 //对象已有必须手动取消的任务，不空闲

#define ACTION\_INVALID\_BUILDACT\_NEEDBUILT -11 //建筑还在建造过程中

#define ACTION\_INVALID\_BUILDACT\_LOCK -13 //建筑行动未解锁，或该行动只能进行有限次且已达上限

#define ACTION\_INVALID\_BUILDACT\_MAXHUMAN -14 //造单位行动，已达人口上限

#define ACTION\_INVALID\_HUMANBUILD\_DIFFERENTHIGH -41 //建筑位置有高度差

#define ACTION\_INVALID\_HUMANBUILD\_OVERBORDER -42//建筑位置加上建筑宽度，超出边界

#define ACTION\_INVALID\_HUMANBUILD\_UNEXPLORE -43 //建筑位置未被探索

#define ACTION\_INVALID\_HUMANBUILD\_OVERLAP -44 //建筑位置与其他物体有重叠冲突

#define ACTION\_INVALID\_HUMANBUILD\_LOCK -45 //建筑未解锁，未达成建筑条件

#define ACTION\_INVALID\_DISTANCE\_FAR -99 //距离相距太远

#define ACTION\_INVALID\_FULLY\_LOAD -119 //携带人物满了

#define ACTION\_INVALID\_POSITION\_NOT\_FIT -520 //建筑放置的位置不合适

6、资源类型常量（tagResource的type成员）

#define RESOURCE\_EMPTY 0

#define RESOURCE\_TREE 60

#define RESOURCE\_STONE 21

#define RESOURCE\_GAZELLE 71

#define RESOURCE\_ELEPHANT 72

#define RESOURCE\_LION 73

#define RESOURCE\_GOLD 99

#define RESOURCE\_FISH 119

7、Farmer采集的资源类型常量（tagResource的ProductSort成员）

#define HUMAN\_WOOD 1 //木头

#define HUMAN\_STOCKFOOD 2 //可放入仓库的食物

#define HUMAN\_STONE 3 //石头

#define HUMAN\_GOLD 4 //黄金

#define HUMAN\_GRANARYFOOD 5 //可放入谷仓的食物

#define HUMAN\_DOCKFOOD 6 //可放入船坞的食物

8、其他可能用到的资源常量

//血量

#define BLOOD\_TREE 25

#define BLOOD\_GAZELLE 8

#define BLOOD\_ELEPHANT 45

#define BLOOD\_LION 20

#define BLOOD\_FOREST 100

//资源总量

#define CNT\_TREE 75

#define CNT\_GAZELLE 150

#define CNT\_ELEPHANT 300

#define CNT\_LION 100

#define CNT\_UPGRADEFARM 325

#define CNT\_BUSH 150

#define CNT\_STONE 250

#define CNT\_GOLDORE 200

#define CNT\_FOREST 300

#define CNT\_FISH 200

9、村民的nowState状态常量

#define HUMAN\_STATE\_IDLE 0 //空闲状态

#define HUMAN\_STATE\_WALKING 1 //正在移动状态（无目标对象）

#define HUMAN\_STATE\_WORKING 2 //正在工作状态（包括打猎）

#define HUMAN\_STATE\_ATTACKING 3 //正在攻击敌军状态

## 5. AI代码示例

1、控制对象移动

tagInfo info;

void UsrAI::processData()

{

//每一帧控制列表第一个空闲的农民移动，

//注意列表内元素顺序是随机的，所以每一帧控制的对象可能不同

    info = getInfo();

    for (tagFarmer& farmer : info.farmers) {

        //农民类型为陆地生产单位，状态为空闲

        if (farmer.FarmerSort == 0 && farmer.NowState == HUMAN\_STATE\_IDLE) {

            int x = 72, y = 72; //假设是一个可达的块坐标

            //命令农民移动到(x,y)，命令接受的是细节坐标，需要转换

//如果对象是运输船，且x,y是一个合法的靠海的陆地，则运输船到目的地后会自动放下运载的对象

            HumanMove(farmer.SN, x \* BLOCKSIDELENGTH, y \* BLOCKSIDELENGTH);

            break;

        }

    }

}

2、控制对象工作

tagInfo info;

void UsrAI::processData()

{

    info = getInfo();

    // 先找到一个空闲的农民

    tagFarmer\* idleFarmer = nullptr;

    for (tagFarmer& farmer : info.farmers) {

        // 农民类型为陆地生产单位，状态为空闲

        if (farmer.FarmerSort == 0 && farmer.NowState == HUMAN\_STATE\_IDLE) {

            idleFarmer = &farmer;

            break;

        }

    }

    // 如果有空闲农民，再找资源

    if (idleFarmer != nullptr) {

        for (tagResource& resource : info.resources) {

            // RESOURCE\_TYPE可以改为RESOURCE\_STONE、RESOURCE\_GAZELLE、RESOURCE\_GOLD等

            // 分别对应执行挖石头、打猎羚羊、挖金矿等动作

            // 这里以砍树为例

            if (resource.Type == RESOURCE\_TREE) {

                HumanAction(idleFarmer->SN, resource.SN);

                break;

            }

        }

    }

}

3、控制对象建造建筑

tagInfo info;

void UsrAI::processData()

{

    info = getInfo();

    for (tagFarmer& farmer : info.farmers) {

        //农民类型为陆地生产单位，状态为空闲，木头足够

        if (farmer.FarmerSort == 0 && farmer.NowState == HUMAN\_STATE\_IDLE && info.Wood >= BUILD\_HOUSE\_WOOD) {

            int x = 72, y = 72; //假设以该点为左下角的2\*2区域是一个合法的区域

            HumanBuild(farmer.SN, BUILDING\_HOME, x, y);

            break;

        }

    }

}

4、控制建筑执行动作

tagInfo info;

void UsrAI::processData()

{

    info = getInfo();

    for (tagBuilding& building : info.buildings) {

        //如果市镇中心空闲，且有足够的食物，且农民数量小于人口上限，则生产农民

        if (building.Type == BUILDING\_CENTER && building.Project == 0 && info.Meat >= BUILDING\_CENTER\_CREATEFARMER\_FOOD &&

            info.farmers.size() < info.Human\_MaxNum) {

            BuildingAction(building.SN, BUILDING\_CENTER\_CREATEFARMER);

            break;

        }

    }

}

5、控制对象上运输船

tagInfo info;

void UsrAI::processData()

{

    info = getInfo();

    // 先找到一个空闲的运输船

    tagFarmer\* idleShip = nullptr;

    for (tagFarmer& farmer : info.farmers) {

        // 单位类型为运输船，状态为空闲，未满载

        if (farmer.FarmerSort == 1 && farmer.NowState == HUMAN\_STATE\_IDLE && farmer.Resource < 5) {

            idleShip = &farmer;

            break;

        }

    }

    // 如果有空闲运输船，再找空闲农民

    // 假设运输船可达

    if (idleShip != nullptr) {

        for (tagFarmer& farmer : info.farmers) {

            // 命令所有空闲的农民上船

            if (farmer.FarmerSort == 0 && farmer.NowState == HUMAN\_STATE\_IDLE) {

                HumanAction(farmer.SN, idleShip->SN);

            }

        }

    }

}

6、控制军队攻击敌军

tagInfo info;

void UsrAI::processData()

{

    info = getInfo();

    // 先找到一个敌军

    tagArmy\* enemy\_army = nullptr;

    for (tagArmy& army : info.enemy\_armies) {

        // 这里假设目标定为敌方战船

        if (army.Sort == AT\_SHIP) {

            enemy\_army = &army;

DebugText("找到敌军战船");

            break;

        }

    }

    // 如果有敌军战船，再找我方战船

    if (enemy\_army != nullptr) {

        for (tagArmy& army : info.armies) {

            // 命令所有空闲的战船攻击敌军战船

            if (army.Sort == AT\_SHIP && army.NowState == HUMAN\_STATE\_IDLE) {

                HumanAction(army.SN, enemy\_army->SN);

                break;

            }

        }

    }

}

7、记录指令id，通过返回值查看错误信息

int id = -1; //使用全局变量记录指令id

tagInfo info;

void UsrAI::processData()

{

    info = getInfo();

    if (id != -1) {

        cout << "id: " << id << " 返回值: " << info.ins\_ret[id] << endl;

        // 输出-3，表示指定位置越界

        id = -1; // 重置id

    }

    for (tagFarmer& farmer : info.farmers) {

        if (farmer.NowState == HUMAN\_STATE\_IDLE) {

            // 记录指令id，在下一次调用processData时查看返回值

            id = HumanMove(farmer.SN, 256 \* BLOCKSIDELENGTH, 256 \* BLOCKSIDELENGTH);

            break;

        }

    }

}

8、综合示例：简单盖一座房屋

tagInfo info;

int begin\_x = 0, begin\_y = 0;

int center\_x = 0, center\_y = 0;

bool home\_flag = false;

// 从市镇中心开始查找2\*2的空地，通过引用返回空地左下角坐标x,y

int find\_block(int& x, int& y) {

    for (int i = begin\_x + 1; i < 128; i++) {

        for (int j = begin\_y + 1; j < 128; j++) {

            if (i + 1 < 128 && j + 1 < 128) {

                // 检查四个位置是否都是陆地

                if ((\*info.theMap)[i][j].type == MAPPATTERN\_GRASS &&

                    (\*info.theMap)[i + 1][j].type == MAPPATTERN\_GRASS &&

                    (\*info.theMap)[i][j + 1].type == MAPPATTERN\_GRASS &&

                    (\*info.theMap)[i + 1][j + 1].type == MAPPATTERN\_GRASS) {

                    cout << "找到陆地: " << i << ", " << j << endl;

                    // 检查高度是否相等（平地）

                    int height = info.theMap[i][j]->height;

                    if ((\*info.theMap)[i + 1][j].height == height &&

                        (\*info.theMap)[i][j + 1].height == height &&

                        (\*info.theMap)[i + 1][j + 1].height == height) {

                        // 找到符合条件的2\*2空地，返回左下角坐标

                        x = i;

                        y = j;

                        cout << "找到空地: " << x << ", " << y << endl;

                        return 1; // 返回1表示成功找到

                    }

                }

            }

        }

    }

    return 0; // 返回0表示未找到合适的空地

}

void UsrAI::processData()

{

    info = getInfo();

    // 初始化市镇中心坐标

    if (center\_x == 0 && center\_y == 0) {

        for (tagBuilding& building : info.buildings) {

            if (building.Type == BUILDING\_CENTER) {

                center\_x = building.BlockDR;

                center\_y = building.BlockUR;

            }

        }

    }

    // 初始化搜索的起始坐标

    if (begin\_x == 0 && begin\_y == 0) {

        begin\_x = center\_x;

        begin\_y = center\_y;

    }

    // 检查是否已经建造了房屋

    for (tagBuilding& building : info.buildings) {

        if (building.Type == BUILDING\_HOME) {

            home\_flag = true;

            break;

        }

    }

    int home\_x, home\_y;

    // 如果未建造房屋，且找到空地，则建造房屋

    if (!home\_flag && find\_block(home\_x, home\_y)) {

        for (tagFarmer& farmer : info.farmers) {

            if (farmer.NowState == HUMAN\_STATE\_IDLE) {

                // 这里未判断该地区是否有其他对象占用，若占用则建造失败，下一帧从失败处开始搜索

                // 建议自行创建一张全局的资源图，记录每一格上的对象

                HumanBuild(farmer.SN, BUILDING\_HOME, home\_x, home\_y);

                break;

            }

        }

    }

    // 记录下房屋坐标，用于后续搜索的起始坐标

    begin\_x = home\_x;

    begin\_y = home\_y;

}

9、综合示例：简单任务转换

tagInfo info;

void UsrAI::processData()

{

    info = getInfo();

    int treeSN = -1;

    int stoneSN = -1;

    int animalSN = -1;

    int fishSN = -1;

    for (tagResource& resource : info.resources) {

        if (resource.Type == RESOURCE\_TREE) {

            treeSN = resource.SN;

        }

        else if (resource.Type == RESOURCE\_STONE) {

            stoneSN = resource.SN;

        }

        else if (resource.Type == RESOURCE\_GAZELLE) {

            animalSN = resource.SN;

        }

        else if (resource.Type == RESOURCE\_FISH) {

            fishSN = resource.SN;

        }

    }

    int SNs[4] = { treeSN, stoneSN, animalSN, fishSN };

    int i = 0;

    // 先让农民去收集各类资源

    for (tagFarmer& farmer : info.farmers) {

        if (farmer.NowState == HUMAN\_STATE\_IDLE && farmer.FarmerSort == 0) {

            HumanAction(farmer.SN, SNs[i]);

            i++;

        }

    }

    // 1000帧后，让砍树和挖石头的人去打猎

    if (info.GameFrame > 1000) {

        for (tagFarmer& farmer : info.farmers) {

            if (farmer.NowState == HUMAN\_STATE\_WORKING && farmer.FarmerSort == 0) {

                if (farmer.WorkObjectSN == treeSN || farmer.WorkObjectSN == stoneSN) {

                    HumanAction(farmer.SN, animalSN);

                }

            }

        }

    }

}

注意，以上所有代码仅作为最基本的接口使用示例，实际编程时应该考虑更全面的条件，增加代码的健壮性和可复用性。

## 6. 获得游戏胜利的基本步骤

本次游戏地图分为两块大陆，中间有海洋隔开。玩家出现在较小的大陆上，为已探索状态，有木材、石头、猎物、近岸渔场资源。海上有较多的渔场资源，但处于未探索状态。敌方大陆有木材、黄金、猎物资源，处于未探索状态，其中黄金资源是本游戏的收集目标。海上有敌方战船巡逻，敌方大陆的主要黄金资源由一群士兵和几个箭塔看守。因此玩家需要采集资源、升级时代，建立海军打通海道并登陆敌方大陆，建立足够的军队击败看守黄金的敌军，才能顺利采集到黄金。为简化编程控制过程，可将游戏分为以下几个阶段，在不同的阶段调用不同的代码。

第一阶段：控制村民在岸边渔场捕鱼、捕猎羚羊以快速获取食物，同时安排部分村民砍伐树木、开采石头、建造房屋等基础设施，确保资源平衡，尽快升级时代。

第二阶段：升级各项科技；利用船坞批量生产战船，组建海军部队，对周边海域展开探索，歼灭敌方海军力量，掌控海洋制权，并侦察到敌方大陆的位置。

第三阶段：在敌方大陆登陆后，可选择就地建造兵营生产军队，或通过运输船将本土训练的军队运送至敌方大陆，稳步推进侦察，寻找金矿矿脉（通常位于大陆深处），配合军事打击，削弱并歼灭敌方陆地部队。

第四阶段：待敌方大陆局势相对稳定后，集中大量运输船，将村民批量运送至敌方大陆，快速采集金矿资源。

## 7. 编写AI代码的注意事项：

* UsrAI::processData()是系统调用AI代码的入口。注意这个调用不是一次性的，而是每帧会尝试调用一次。游戏核心本身是按时间发起的事件处理机制，UsrAI::processData()函数实际上是一个会定时执行的循环体。编程时必须考虑到这个因素，至少有两个后果：a.几乎所有的游戏动作都需要有控制条件，不然就成了每帧都发起，会达不到预期；b.控制动作、记录中间过程用的变量，都必须是静态或者全局的，使用局部变量进入代码会被重新初始化造成错误。
* 发出控制命令成功，不等于命令可以执行成功。例如命令村民建造建筑时，如果目标位置上有障碍物（包括单位），则不能建造，该村民的状态会回到HUMAN\_STATE\_IDLE。若目标对象不可达，村民可能会持续寻路尝试靠近目标，此时会一直保持非IDLE状态，需要自己根据情况写代码处理该状态。示例代码中用了其他方法判定村民有无进行命令动作，也是有效的。
* 本系统采用了多线程，AI代码和游戏主线程在时间上可以看作是并行的。AI代码的执行过程不影响主线程的进度，可以执行复杂度较高的代码。但是时间过长会延缓发出指令的时间，有可能会造成指令和期待有差异，因为发指令时的游戏局势已经变化了。建议AI代码每轮执行的时间控制在几帧以内。
* 原游戏的节奏较慢，完成一次任务的时间较长，因此系统提供了倍速功能。在界面右上方有四个可以点击的圆点，分别表示游戏运行在1、2、4、8倍速(8倍速时无法使用鼠标操作)。我们在实际批改运行时，为节省运行时间将使用8倍速运行，同学们在设计和测试AI代码时也要考虑这一因素，最终的测试应当在8倍速下完成。
* 游戏调试时可以使用Qt的debug模式编译，方便动态调试，但代码性能较低。我们在最终测试时会使用release模式编译，提高测试性能。同学们提交代码前也要自己用release模式编译测试。
* UsrAI::processData()中，所有语句都可以重写。但重写在UsrAI.cpp以外的代码无效，最终提交作业只提交UsrAI.cpp和UsrAI.h。建议多设计一些功能函数作为UsrAI的成员函数，分散processData()的压力，增加代码条理；多个成员函数需要共同使用的变量新增为UsrAI的数据成员。新增的UsrAI成员函数和成员变量需要在UsrAI.h中声明。
* 游戏进行时，主线程会在右侧的文字窗口显示指示文字，包括游戏的进度和对AI命令的响应。AI线程也可以使用此窗口显示信息用于调试，使用方法见前述AI示例代码。调试时也可使用QT的输出窗口，建议将临时输出的内容输出到QT窗口，重要的调试信息输出到游戏的文字窗口。
* 本任务较为复杂，按要求认真完成代码量较大，同学们设计代码时应当多使用函数或类（类在提交时，代码必须并入UsrAI.cpp），使代码更有条理、更简短。代码中用到常数的地方尽量使用#define定义，以增强可读性和方便调试。有能力的同学可以设计通用的调度系统调度农民和军事单位，以简化编程，提高代码效率。