|  |
| --- |
| 沈阳航空航天大学计算机博弈协会 |
| **SAU Game Platform** |
| 开发记述文档 |
|  |
| **尹伟和** |
| **2014/7/15** |

|  |
| --- |
| © 2014 沈阳航空航天大学计算机博弈协会 |

目录

[第1章 需求分析 1](#_Toc400395636)

[1.1 研发背景 1](#_Toc400395637)

[1.2 研发定位 1](#_Toc400395638)

[1.3 基本需求 1](#_Toc400395639)

[1.4 扩展功能 2](#_Toc400395640)

[1.5 易用性分析 2](#_Toc400395641)

[1.6 系统环境 2](#_Toc400395642)

[第2章 架构概要设计 3](#_Toc400395643)

[2.1 整体架构 3](#_Toc400395644)

[2.2 对弈逻辑控制模块设计 4](#_Toc400395645)

[2.3 棋种支持模块设计 5](#_Toc400395646)

[2.4 博弈引擎控制模块设计 6](#_Toc400395647)

[2.5 网络接口模块设计 7](#_Toc400395648)

[2.6 图形用户界面设计 7](#_Toc400395649)

[2.7 计时器模块设计 8](#_Toc400395650)

[2.8 扩展功能支持模块设计 9](#_Toc400395651)

[2.9 操作控制模块设计 9](#_Toc400395652)

[第3章 关键技术细节 10](#_Toc400395653)

[3.1 平台-引擎通信协议 10](#_Toc400395654)

[3.1.1 数据格式 10](#_Toc400395655)

[3.1.2 命令字 10](#_Toc400395656)

[3.1.3 棋盘坐标 10](#_Toc400395657)

[3.1.4 平台向引擎发送命令串定义 10](#_Toc400395658)

[3.1.5 引擎向平台发送命令串定义 13](#_Toc400395659)

[3.1.6 其它 13](#_Toc400395660)

[3.2 裁判扮演者——对弈逻辑控制模块 14](#_Toc400395661)

[3.2.1 裁判对棋手的选择与对弈模式设定 14](#_Toc400395662)

[3.2.2 棋手行棋事件获取 15](#_Toc400395663)

[3.2.3 裁判行为通知 16](#_Toc400395664)

[3.2.4 引擎棋手的虚拟角色 17](#_Toc400395665)

[3.2.5 强制终止对弈逻辑 17](#_Toc400395666)

[3.3 博弈平台的基础——棋种支持模块 18](#_Toc400395667)

[3.3.1 接口函数定义规范 18](#_Toc400395668)

[3.3.2 实现棋种支持模块的说明 23](#_Toc400395669)

[3.4 底层驱动——博弈引擎控制模块 24](#_Toc400395670)

[3.4.1 双缓冲技术获取引擎发送命令串 24](#_Toc400395671)

[3.5 网络通信控制协议 24](#_Toc400395672)

# 需求分析

## 研发背景

今天，计算机博弈在中国大陆高校范围已经有了初步发展，全国大学生计算机博弈大赛业已成功举办了四届。然而，当两名开发人员各执自己的博弈引擎程序进行对战或是研发人员自行测试自己引擎的功能时，手工输入命令十分不便，也有可能因为一次错误的坐标输入导致一局对弈作废。正因如此，亟需研发一款可以实现引擎之间对弈自动化的博弈平台软件，将人力从繁琐的操作中解放出来。

另外，对于刚刚开始从事博弈引擎开发的大学生来说，匮乏的GUI编程经验使得他们不得不花费精力在图形化界面的实现上，而不能专注于博弈引擎性能的提升。因此，开发一款具有友好的用户界面的平台软件作为博弈引擎的图形界面扩展也是十分必要的。

## 研发定位

基于上述原因，我们决定开发一款能够实现博弈引擎对弈自动化的，具有友好用户界面的，易于功能扩展的通用博弈平台——SAU Game Platform。

## 基本需求

1. 在平台-引擎通信协议框架下，博弈平台程序可以与博弈引擎程序进行进程间通信，实现引擎对弈自动化，即引擎间对弈。
2. 提供网络接口，实现网络上两个平台副本之间的对弈功能，满足用户对更多的计算机资源的需求。
3. 提供用户进行手动行棋功能，实现包括人人对弈、人机对弈等功能，方便用户测试博弈引擎性能。
4. 提供友好的界面，以弥补博弈引擎开发者在GUI编程上的资源损耗，使开发者可以将更多的精力专注于对博弈引擎性能的提升上。
5. 在用户参与的对弈模式中，引入着法确认与取消机制来规避因为错误的着法输入带来的对弈作废等恶劣影响。
6. 博弈平台要实现多棋种支持，以体现其通用性。

## 扩展功能

1. 对双方棋手行棋进行计时。
2. 多语言风格支持。
3. 记录并保存对弈信息（招法历史）。
4. 对棋盘进行截图，以图形化的方式保存对弈信息。
5. 提供对外部工具程序的调用入口。
6. 实现对保存于程序指定目录下（包括文件夹下）的棋谱、截图的删除。
7. 实现对程序背景、棋盘材质、背景音乐等的选则与控制

## 易用性分析

1. 对于棋种支持组件、语言资源组件、外部工具程序入口等，应实现动态整合，易于程序功能的扩展与移除。
2. 对于行棋过程的逻辑模拟，应该做到与具体棋种无关，以保证棋种组件的接口规范化。
3. 提供良好的定义明确的接口，以便于第三方协助开发，为软件的推广创造良好的条件。

## 系统环境

考虑到目前国内主流操作系统是Microsoft开发的Windows NT内核操作系统，软件指定在Windows NT系列操作系统上运行，采用C/C++语言进行开发，使用标准模板库(STL,Standard Template Library)作为部分底层数据结构的实现，至于与操作系统的交互则采用Windows操作系统提供的Win32 API应用程序接口。

# 架构概要设计

## 整体架构

在博弈活动中有两种身份的博弈参与者，其一是主导博弈活动进行的棋手，其二是维护整个博弈活动正常有序进行的裁判。博弈活动可以看成：棋手在裁判定义的规则框架下，根据裁判指示完成的有序行为序列的统一。SAU Game Platform作为博弈活动参与者的身份定位就是裁判，而自然人或者博弈引擎作为棋手在其调度下完成博弈活动。

如图1所示，博弈平台的架构包括博弈逻辑控制模块(Game Logic Control)、棋种支持模块(Board Game Support)、博弈引擎控制模块(Game Engine Control)、网络接口模块(Network Interface)、图形用户界面(Graph User Interface)、计时器模块(Timer) 、扩展功能支持模块(Extended Function Support) 和操作控制模块(Operational control)八个部分。



图 1 SAU Game Platform架构图

其中构架的核心是博弈逻辑控制模块，该模块是博弈平台裁判身份的直接体现，控制对弈逻辑，从全局的高度统筹博弈活动的进行。构架的基础棋种支持模块，该模块是对特定于棋种的实现的封装，包括棋种规则定义实现、棋盘绘制、棋局状态维护和平台行为控制四部分。博弈引擎控制模块用来实现平台与引擎之间的通信，而网络接口模块则用来实现平台与网络另一端平台副本之间的通信。

图形用户界面是平台与用户进行直接交互的部分，强调用户体验，在简洁、友好的界面风格基础上强调程序的实用性和易用性。计时器模块用来记录棋手行棋耗时。扩展功能支持模块则实现了使程序内容变得丰富的元素，包括图片选择、背景音乐、保存棋谱截图等，提高程序的个性化和功能的多样性。

最后，操作控制模块是程序功能接口的汇集，上接图形用户界面，下连程序其他模块，用来将用户操作分派到各个模块中进行实现。

## 对弈逻辑控制模块设计

对弈逻辑控制模块作为SAU Game Platform构架的核心部分，是整个博弈活动的总控制模块，提供对真实博弈过程的模拟实现，控制博弈引擎接口和网络接口在整个博弈过程中的IO调度，并根据棋种支持模块中提供的棋种规则指导棋手的行为。

对弈逻辑控制模块是在对众多棋类博弈逻辑过程进行高度抽象的基础上，进过归纳总结而产生的逻辑控制单元。博弈活动的参与者是两种身份不同的角色，即棋手和裁判。在SAU Game Platform架构体系中，博弈引擎、网络另一端的平台副本或用户扮演棋手角色，逻辑控制模块则扮演博弈活动的裁判角色。

对博弈活动的抽象假设：裁判是博弈规则的制定者和博弈活动的监督者，棋手的行为是对裁判发出的指示的响应。平台中进行的整个博弈活动都是受对弈逻辑控制模块统筹控制的，博弈引擎、网络另一端的平台副本和用户所扮演的棋手角色的行为是根据对弈逻辑控制单元的指令而产生的响应行为，而对弈逻辑控制单元的后续指令又是对棋手反馈的行为的响应。因此，对弈逻辑控制模块中采用行棋事件驱动模型(Move Event Drive)来实现对现实博弈逻辑过程的抽象模拟。

什么是事件？在对弈过程中，棋手对棋局状态产生影响的一次有效操作即为一个事件。

因为棋种支持模块将完全实现棋种规则定义，只要SAU Game Platform获知棋手的行棋操作，就可以自主维护棋局状态并产生后续的裁判角色行为。因此，棋手的行棋事件完全可以驱动对弈逻辑的顺次执行，使博弈活动正常进行。

对弈逻辑是一个循环过程，其在对弈逻辑控制模块指导下是这样执行的：

对弈开始时，若有博弈引擎或网络另一端的平台副本参与博弈活动，则平台向其发送对弈开始命令，并分配棋手的角色——黑方或白方。然后对弈逻辑的执行进入暂停状态，等待棋手发出行棋事件。当对弈逻辑控制模块获取到棋手的行棋事件时，模块将行棋事件交付给棋种支持模块进行处理，并根据棋种支持模块返回的裁判角色行为指导棋手的下一步行动。此时，若已分胜负则对弈逻辑控制模块终止对弈，否则暂停对弈逻辑的执行并等待下一次行棋事件。

对弈逻辑控制模块指导下进行的博弈活动在时间上是连续的，在过程上是分阶段的、离散的。为了保障博弈活动正确、完好地进行下去，对弈逻辑控制模块需要对对弈逻辑的阶段进行编码。当博弈者行为触发对弈逻辑的执行时，对弈逻辑控制模块将根据阶段编码判断博弈者的行为是否符合阶段需要，进而决定接受或忽略该行为，并由此保证对弈逻辑不会因为博弈者的非正常行为而陷入混乱之中。

## 棋种支持模块设计

棋种支持模块是SAU Game Platform构架的基础部分，在其中完成棋种规则定义实现、棋盘绘制、棋局状态维护和裁判角色行为控制四项工作。针对不同棋种，棋种支持模块具有不同的实现，同时考虑到多棋种支持的实用性和棋种模块增添、移除的易用性，决定采用动态链接库(DLL,Dynamic-Link Library)作为棋种支持模块的实现载体。

棋种支持模块定义一套棋种无关的、通用的标准接口函数，提供博弈平台程序与动态链接库之间的交互。SAU Game Platform为每个棋种编写一个独立的动态链接库，这样博弈平台只需在运行过程中载入不同的动态链接库，并获取其中的标准接口函数地址，就可以实现不同棋种支持之间的切换。棋种支持模块增添、移除的易用性则体现为：只需要编写新的动态链接库或删除已有的动态链接库就可以添加对新棋种的支持或移除对已支持棋种的支持，而不需要对程序进行重新编译。

在棋种支持模块的四项工作中，棋盘绘制依赖于棋局状态，棋局状态维护依赖于棋种规则定义，裁判角色行为控制受控于棋种规则定义和棋局状态。

在对弈逻辑执行过程中，棋种支持模块接收到对弈逻辑控制模块发送的行棋事件后，首先根据协议约定将行棋事件解析成着法，然后根据棋种规则的定义判断着法的正确性，并根据着法的正确与否决定接受或拒绝该着法和产生相应的裁判角色行为返回给逻辑控制模块。

需要说明的是：在自然人作为棋手的对弈模式下，由于引入了着法确认与取消机制，用户输入的着法并不会作为行棋事件驱动对弈逻辑继续执行，直到用户对着法进行确认才会真正产生行棋事件。虽然用户着法不会立即生成行棋事件，但用户应该可以直接观察到该着法的效果，即用户输入的着法仍会即时地改变棋盘的内容。因此，在棋种支持模块中对用户输入着法的处理不应是直接改变棋局状态，而应是进行暂存，并等到用户对该着法进行确认后再改变棋局状态。

## 博弈引擎控制模块设计

博弈引擎控制模块是博弈平台与博弈引擎程序的接口，负责所有对博弈引擎程序的直接操作，包括与博弈引擎的连接、断开和数据读写。

在整个博弈活动中，博弈引擎控制模块承担的工作核心是完成本地计算机上运行的博弈平台程序与博弈引擎程序之间的数据交换，其核心技术是进程间通信(IPC,Inter-Process Communication)。Windows系统环境下实现进程间通信的方法有很多，但考虑到引擎程序编写的简便性和实用性，最终决定使用消息管道(Message Pipe)来实现博弈平台与引擎间的进程间通信。为了满足不同类型应用程序的需要，SAU Game Platform同时实现匿名管道([Anonymous pipes](ms-help://MS.MSDNQTR.2003JAN.1033/ipc/base/anonymous_pipes.htm))和命名管道([Named pipes](ms-help://MS.MSDNQTR.2003JAN.1033/ipc/base/named_pipes.htm))两种进程间通信机制，并可由用户按需选择。

在命名管道和匿名管道两种进程间通信机制中，匿名管道是专门为控制台程序提供。使用匿名管道对控制台程序的标准输入输出进行重定向不仅可以简便地实现进程间通信，而且对控制台形式的博弈引擎程序具有良好的通信技术细节隐藏。此时，引擎既不必关心输入是来自用户还是来自平台程序，又不必关心输出是经过屏幕呈现在用户的视网膜上还是被平台获取，因为两种形式都是通过标准的输入输出来完成的。

命名管道则是用来提供给Win32应用程序来连接博弈平台的。由于命名管道需要博弈引擎程序主动进行连接和读写管道中的数据，所以编写使用命名管道连接博弈平台的博弈引擎程序相对来说要困难很多。

在博弈引擎程序作为棋手的对弈模式中，需要SAU Game Platform加载引擎程序，建立平台与引擎之间的连接关系。若从全局的高度考察这一引擎加载过程，其应该归到对弈逻辑控制模块下，但其核心操作则是由博弈引擎控制模块下的引擎连接过程来完成的。在引擎连接过程中，博弈引擎控制模块首先建立用来通信的消息管道，再启动引擎程序子进程，最后等待引擎程序连接消息管道。

与引擎连接过程相似，引擎断开过程负责完成对弈逻辑控制模块下引擎卸载过程的主要操作。在引擎断开过程中，首先通知引擎断开消息管道连接，再关闭引擎进程，最后释放消息管道。

数据读写过程则是调用Windows系统提供的文件读写函数——ReadFile和WriteFile——来完成管道数据的读写。

## 网络接口模块设计

网络接口模块是博弈平台进行网络通信的接口，负责与其他博弈引擎程序副本和服务器程序之间的通信。SAU Game Platform架构中的网络传输以Winsock WASAsyncSelect（异步选择）模型进行实现，传输数据的格式受到软件定义的网络通信控制协议约束。

在对弈逻辑执行过程中，网络接口模块在两个平台副本的对弈逻辑控制模块之间架设一条数据传输通道，用来传递行棋事件，并以此驱动对弈逻辑的执行。用户对于网络功能的操作以及网络两端的用户之间的会话，则由图形窗口进行辅助控制。

## 图形用户界面设计

图形用户界面是SAU Game Platform架构中直接与用户进行交互的部分，是整个程序的外部包装，直接影响用户体验。在整个图形用户界面设计中，应坚持简洁、实用的风格，给用户最佳的使用体验。

SAU Game Platform的图形用户界面由主体棋盘、操作面板、菜单栏和状态栏四个部分组成。

主体棋盘包括：棋盘图像、背景图片和人物图片三种图像元素。其中棋盘图像直接向用户呈现棋局状态信息，同时也接受用户的行棋操作，背景图盘和人物图片则是用来烘托对局的氛围以给用户更好的体验。同时考虑到不同的棋种对于棋盘内容有不同的表示，棋盘图像被划分为前景和背景两个层次。棋盘图像的前景用来显示棋盘的刻线和其上的棋子，由棋种支持模块负责绘制；棋盘图像的背景则是一张用来反映棋盘材质的图片，其目的是为了凸显棋盘图像的前景内容。

操作面板用来显示对弈的关键信息，并提供一些常用操作的便捷按钮。操作面板被划分为三部分，由两个分属对弈双方的组合框和历史着法列表组成。其中每个组合框中包括显示引擎名（默认为Person）、行棋计时两种信息的文本，和加载引擎、卸载引擎、确认着法、取消着法四种操作按钮。历史着法列表则是提供双方棋手着法的文本显示的文本框。

对用户来说，菜单栏是SAU Game Platform各种功能的主要提供者，通过在各个菜单项与操作控制模块提供的各种功能接口之间建立映射关系来实现用户操作的分派。

最后，状态栏用来显示SAU Game Platform提供的各种功能的解释说明，并显示程序当前所处的状态的说明。

## 计时器模块设计

计时器模块用来提供对棋手行棋时耗的记录，并显示在图形用户界面中。

为了使计时效果更为精确，SAU Game Platform中设计了一种名为“分时计时”的策略来进行细粒度的时间计算。分时计时的原理是这样的：

首先，计时器记录对弈开始的时间，并将双方棋手行棋时耗置零。然后计时器在一方棋手产生行棋事件时进行计时的更新。在进行计时更新时，计时器先获取当前的时间，并用其减去对弈开始时间从而得到对弈的持续时间。再之后，计时器将用对弈持续时间减去对方棋手的行棋时耗，从而得到当前行棋棋手的绝对行棋时耗。

当然，为了在图形用户界面上显示动态的计时效果，计时器将设置系统定时器，以一定的节拍进行计时更新，并进行相应部分的重绘。

## 扩展功能支持模块设计

扩展功能支持模块并非一个独立的功能模块，而是那些使程序内容变得丰富的功能的集合。这些功能包括：设置背景、设置棋盘材质、设置人物图片、保存棋谱、保存棋盘截图、播放背景音乐等，它们的实现渗透在其它各个模块中。

## 操作控制模块设计

操作控制模块是对程序提供各项功能的接口汇总。

在操作控制模块中，其它各模块中提供的那些以参数作为执行策略的通用功能函数接口将被展开，形成以无参函数形式实现指定单一功能的标准通用接口。只有无参函数形式的功能入口，才能以统一类型的函数指针与菜单项形成映射关系，并以普适的形式进行调用。

此外，针对用户行棋这一独特的用户点击操作，操作控制模块将提供额外的可以传递指针坐标的函数入口。

# 关键技术细节

## 平台-引擎通信协议

平台-引擎通信协议是SAU Game Platform博弈平台程序与博弈引擎程序之间进行通信的数据格式规范，是实现博弈平台与博弈引擎间通信的实际载体。只有双方共同遵守该协议，双方才能对传递的数据进行解析，获得数据的实际意义。

### 数据格式

平台-引擎通信协议规定：博弈平台与博弈引擎间通信的数据以字符串的形式进行传递，有效信息是 “command word+[‘ ’+parameters]+‘\n’” 格式的命令串，即以命令字开头换行符结尾中间填充可选参数，而无效信息将被自动过滤。

平台-引擎通信协议规定：博弈平台与博弈引擎间通信的一条命令串长度不超过256字节（包括字符串结束符‘\0’）。

### 命令字

平台-引擎通信协议定义有：“name?”, “name”, “new”, “move”, “take”, “error”, “end”七个命令字。其中 “name?”, “new”, “move”, “error”, “take”, “end”为平台向引擎传递命令字，“name”, “move”为引擎向平台传递命令字。

在后面对命令串进行完整定义的小结中，以 “[” 和 “]” 括起的参数为可选参数。

### 棋盘坐标

平台-引擎通信协议建议：二维棋盘分为横轴和纵轴两个维度，以左上角为坐标原点，坐标从字母 ‘A’ 开始，按字母序进行标定。

### 平台向引擎发送命令串定义

1. “name?” 命令

功能：

询问引擎名称。

完整格式：

“name?+‘\n’”

参数：

无。

说明：

“name?” 命令为无参命令，只能由平台向引擎发送。

1. “new” 命令

功能：

创建对弈。

完整格式：

“new+‘ ’+side+‘\n’”

参数：

side：用于指明引擎的执棋颜色，其值为black或white。

说明：

“new” 命令用于通知引擎创建新棋局。若side值为black，则引擎先手行棋，应向平台发送“move”命令。若side值为white，则引擎后手行棋，应等待引擎发送行棋通知，即“move”命令。

1. “move” 命令

功能：

行棋通知。

完整格式：

“move+[‘ ’+position list]+‘\n’”

参数：

position list：坐标列表。

说明：

“move” 命令用于通知引擎开始行棋，其参数为上一手棋的着法。坐标列表内坐标按字符位数进行区分，两个坐标之间不加空格符，两个字符构成一个坐标，第一字符表示横轴坐标，第二字符表示纵轴坐标。如幻影围棋类的非完备信息博弈，不可获知对方的行棋着法，则该命令不带参数。添子类棋种坐标列表内坐标数量等同于落子数量，每一个坐标表示一处棋子落点。走子类棋种坐标列表内坐标数量为行棋步数加一，其中第一坐标为棋子的起始坐标，最后坐标为棋子落点坐标，其余坐标为棋子的滞留坐标。如亚马逊棋需要设置障碍，则在坐标列表中添加一个障碍坐标。

1. “take” 命令

功能：

提子通知。

完整格式：

“take+‘ ’+num+‘ ’+position list+‘\n’”

参数：

num：坐标列表内坐标数量。

position list：坐标列表。

说明：

“take” 命令用于通知引擎进行提子。该命令用于幻影围棋类引擎不可推知何处棋子如何被移出棋盘的非完备信息博弈，其它类型博弈的提子或吃子操作皆可通过对 “move” 命令的处理得出。

1. “error” 命令

功能：

通知着法错误。

完整格式：

“error+‘\n’”

参数：

无。

说明：

“error” 命令为无参命令，只能由平台向引擎发送。

1. “end” 命令

功能：

通知对弈结束。

完整格式：

“end+[‘ ’+win]+‘\n’”

参数：

win：获胜方，值为black或white。

说明：

“win” 命令参数可由可无，若含参数，则参数用以指定获胜方。

### 引擎向平台发送命令串定义

1. “name” 命令

功能：

应答引擎名称。

完整格式：

“name+‘ ’+engine name+‘\n’”

参数：

engine name：引擎名称。

说明：

“name” 命令用于应答平台发出的 “name?” 命令。

1. “move” 命令

功能：

应答行棋着法。

完整格式：

“move+‘ ’+position list+‘\n’”

参数：

position list：着法列表。

说明：

“move” 命令用于应答平台发出的 “move” 命令和 “new black” 命令，position list参数规则参考平台向引擎发送的 “move” 命令说明。

### 其它

SAU Game Platform对“move”命令的处理，“move”, “take”, “error”命令的发出是在棋种支持模块中进行的，具体的命令定义及解析过程可由棋种支持模块规定。

## 裁判扮演者——对弈逻辑控制模块

博弈活动是棋手在裁判定义的规则框架下，根据裁判指示完成的有序行为序列的统一。在SAU Game Platform参与的博弈活动中，其扮演角色便是裁判，而这一身份的直接体现者则是对弈逻辑控制模块。因此可以说：整个博弈活动就是在对弈逻辑控制模块的统筹调配下进行的。

### 裁判对棋手的选择与对弈模式设定

博弈活动的进行离不开裁判，同样也离不开棋手。在SAU Game Platform中允许三种形式的棋手，分别是用户、博弈引擎程序和网络另一端的平台副本。这样，裁判应该可以根据棋手的形式自动适配对弈模式。

在SAU Game Platform中，默认双方棋手均为用户，如需博弈引擎程序或网络另一端平台副本扮演棋手角色，则需要向对弈逻辑控制模块进行注册。如此，当用户执行对弈开始操作时，对弈逻辑控制模块先检查棋手角色的形式，并据此选着适当的对弈模式执行对弈逻辑。

对弈逻辑大体上可分五类：人人对弈、人机对弈、机机对弈、人网对弈、机网对弈。不同的对弈模式之间对于基本对弈逻辑的理解并无不同，只是在不同形式的棋手的行棋事件获取和裁判行为通知等交互操作上存在差异。

为了快速的对对弈模式进行识别，SAU Game Platform中设置了GameMode变量用来记录对弈模式，同时设置NetWork变量用来记录是否开启网络对战。GameMode的值及对应含义如下：

-1：对弈尚未开始。

0：用户扮演黑方棋手，博弈引擎或网络另一端平台副本扮演白方棋手。

1：用户扮演白方棋手，博弈引擎或网络另一端平台副本扮演黑方棋手。

2：机机对弈，即两个博弈引擎之间进行对弈。

3：人人对弈。

4：博弈引擎扮演黑方棋手，网络另一端平台副本扮演白方棋手。

5：博弈引擎扮演白方棋手，网络另一端平台副本扮演黑方棋手。

### 棋手行棋事件获取

SAU Game Platform对于行棋事件的获取是按对弈阶段和对弈模式来指定输入源的，这样才能保证对弈逻辑正常有序的执行，而不会因为错误的输入源的介入使对弈秩序产生错乱。

首先来说明SAU Game Platform对于用户扮演棋手的行棋事件的获取。SAU Game Platform对用户输入着法的获取是通过用户使用鼠标指针点击棋盘来进行的，当GameMode值为3时，任何时候对弈逻辑控制模块都接受用户输入；当GameMode值为0时，只有当前行棋方为黑方时对弈逻辑控制模块才接受用户输入；当GameMode值为1是，只有当前行棋方为白方时对弈逻辑控制模块才接受用户输入；GameMode为其他值时，用户输入无效。

另外，由于一些棋种的着法输入是多阶段的，即用户的一次输入不能直接作为着法生成行棋事件，所以需要将多次输入交付到棋种支持模块组合成有效着法，然后由用户对着法进行确认在生成行棋事件，并触发对弈逻辑控制模块对行棋事件的处理。

在GameMode值不为3时，SAU Game Platform中采用了数字上的技巧。这一技巧有这样的前提：规定棋种支持模块中对当前行棋方的维护以0为黑方，以1为白方。这样，当GameMode的值与棋种支持模块中维护的当前行棋方相等时即接受用户输入，否则不接受。

对于博弈引擎程序扮演棋手的行棋事件获取，只需要在其输入有效阶段主动读取消息管道中信息即可。博弈引擎程序扮演棋手的行棋事件输入有效阶段为：当GameMode值为0、2、5时，只有当前行棋方为白方时白方方引擎的输入才有效；当GameMode值为1、2、4时，只有当前行棋方为黑方时黑方引擎的输入才有效。然而开发SAU Game Platform过程中，在尝试过此种方案后，最终选择了另一种“引擎棋手虚拟角色”技术。此技术将在3.2.4小结进行详细描述。

最后来说明对于网络另一端平台副本扮演棋手的行棋事件获取。SAU Game Platform中网络另一端的平台副本对对弈逻辑控制模块是不可见的，只能通过网络接口传递数据，而网络接口是采用Winsock WASAsyncSelect异步选择模型实现的。在异步选择模型中，网络接口对数据的接受是被动的，当数据到来时Winsock产生一条窗口消息触发程序的处理过程。基于此种机制，对弈逻辑控制模块对网络另一端的平台副本所扮演的棋手的行棋事件获取也是被动的，当轮到该棋手行棋时对弈逻辑就处于等待状态，直到网络接口模块接收到网络另一端平台副本发送来得行棋事件再触发处理过程，交付给棋种支持模块进行处理。

当进行网络对弈时，对弈逻辑控制模块并不需要刻意地维护网络另一端平台副本行棋事件期间的对弈逻辑，因为在此期间网络另一端平台副本将维护对弈逻辑，补全本端平台副本的工作。

### 裁判行为通知

对弈逻辑控制模块扮演博弈活动的裁判角色，其行为依赖具体棋种的规则定义。但从一种较高的抽象层次上考察则会发现：当裁判角色获取到棋手的行棋事件后，其裁判行为可拆分为需通知给当前行棋棋手裁判行为和需通知给方棋手裁判行为两部分，分别用于指导双方棋手的行动。

依据上述裁判行为拆分模型，对弈逻辑控制模块接收并处理过棋手发出的行棋事件后，便该通知棋手其应该知晓的裁判行为。针对不同形式的棋手，对弈逻辑控制模块对裁判行为通知的细节具有不同的处理方式。

SAU Game Platform与用户的交互全部是通过图形用户界面进行的，而用户对于棋局信息的获取全是从棋盘图像中得到的，有因为棋种支持模块在处理行棋事件时自行维护棋局状态并更新棋盘图像，所以对弈逻辑控制模块不需要显示地通知用户扮演棋手裁判行为。

SAU Game Platform与博弈引擎程序的交互是通过相互之间传递命令串进行的，棋种支持模块同样支持对命令串的解析，因此当棋种支持模块处理完行棋事件后，将相应的裁判行为写成两组命令串返回给对弈逻辑控制模块，并由对弈逻辑控制模块将他们发送给需要的博弈引擎程序。

SAU Game Platform与网络另一端的平台副本的交互是通过实现网络接口的Winsock传递字符串数据，而命令串也是字符串，故此可以将命令串作为裁判行为的载体，通过Winsock进行传递给网络另一端平台副本的网络接口模块。

如果行棋事件使得对弈逻辑自然终结，即出现终盘状态，GameMode的值将被置为-1。应对于此种状况，棋种支持模块中对行棋事件处理的接口应该返回一个状态码指明已处理的行棋事件的信息。

### 引擎棋手的虚拟角色

将扮演棋手角色的博弈引擎程序看成一个客观实体思考，在其生存期间，博弈引擎共向平台发送两种命令串，即“name”命令和“move”命令，而“name”命令只是在博弈引擎与平台完成连接后，响应“name?”命令时使用。在博弈引擎几乎全部生命周期中，它存在的价值就是向SAU Game Platform发送“move”命令串。因此可以使用一个循环不断的读取博弈引擎发出的数据，并从中提取“move”命令串来收获博弈引擎产生的价值。

当获取到“move”命令后，即得到博弈引擎发出行棋事件，此时再去判断对弈逻辑的状态，维护逻辑执行的秩序。若该行棋事件产生的时间点不符合对弈逻辑的要求，该命令将被忽略，循环继续获取下一条“move”命令，否则进入行棋事件的处理及后续流程。

循环中对行棋事件的处理及后续流程是这样的：首先将行棋事件交付给棋种支持模块进行处理，然后根据对弈模式和对弈逻辑状态将裁判行为通知给适应的棋手，最后判断对弈逻辑是否应该终止。该流程执行完毕，循环同样继续获取下一条“move”命令。

如上所述的循环过程就像对弈逻辑控制模块这个裁判创建里一个虚拟的棋手，裁判不断地向这个虚拟的棋手索要行棋着法，并用棋手给的着法来完成对弈逻辑。

引擎棋手虚拟角色这一技术在对弈逻辑控制模块中是依靠多线程来实现的，当一个博弈引擎程序被加载后，对弈逻辑控制模块便启动一个引擎棋手角色线程。在线程中，首先向引擎程序发送“name?”命令，并获取引擎的应答来完成引擎棋手的注册，然后线程便开始执行循环，实时获取引擎输入，直到引擎被卸载时显示地终止线程。

另外，因为引擎棋手虚拟角色技术的实时性，它便在先天上解决了消息管道因为内部数据过度堆积导致的数据丢失问题。

### 强制终止对弈逻辑

对弈逻辑的强制终止就是由用户主动结束对弈命令。此时对弈逻辑控制模块只需要将GameMode的值置为-1，并通知各个棋手对弈被终止即可。当进行网络对弈时，对弈终止信息将由对弈逻辑控制模块要求网络接口模块按照网络通信控制协议的规定发出。

## 博弈平台的基础——棋种支持模块

棋种支持模块是SAU Game Platform的基础，是SAU Game Platform不可或缺的基本组件，没有了棋种支持模块，SAU Game Platform就不能正常工作。SAU Game Platform的价值体现依赖于那些具有不同规则定义的棋种，棋种支持模块中实现的就是与棋种规则密切相关的部分。然而，为了体现SAU Game Platform的通用性和易用性，保证具有不同规则定义的棋种的支持模块都能够没有迟滞地连接到SAU Game Platform主程序中，这就需要在棋种支持模块上层定义一套与棋种规则无关的、通用的接口。

本节主要讨论棋种模块的接口定义规范和其内部实现的一些注意事项。

### 接口函数定义规范

棋种支持模块需要完成棋种规则定义实现、棋盘绘制、棋局状态维护和裁判角色行为控制四项工作，其中棋盘绘制、棋局状态维护和裁判行为控制是需要与SAU Game Platform进行交互的内容，而这些交互则是通过接口函数来实现的。

棋种支持模块对于接口函数的定义规范如下：

1. CheckMoudle 函数

功能：

棋种支持模块正确性校验。

函数原型：

extern “C” \_\_declspec(dllexport) VOID CheckMoudle(

char \*Info,

char \*ChessName,

int \*ChessType

);

参数：

Info [out]：校验字符串缓存区指针。

ChessName [out]：棋种名称字符串缓存区指针。

ChessType [out]：棋种类型识别码指针，为保留参数。

返回值：

无。

说明：

CheckMoudle 函数用于SAU Game Platform初始化过程中扫描棋种支持模块时的正确性校验，和棋种相关信息的获取。在棋种支持模块中，需要用字符串“SAU\_GamePlatform chessType”填充Info。

1. InitMoudle 函数

功能：

棋种支持模块初始化。

函数原型：

extern “C” \_\_declspec(dllexport) VOID InitMoudle(

HWND hWnd,

GameSet \*gameset

);

参数：

hWnd [in]：SAU Game Platform主窗口句柄。

gameset [in]：保存SAU Game Platform设置信息的GameSet结构的指针。

返回值：

无。

说明：

InitMoudle 函数用于初始化棋种支持模块，当在SAU Game Platform中进行棋种切换时调用此函数，向棋种支持模块传递主窗口句柄和程序设置信息。

1. OnSize 函数

功能：

设置棋盘位置。

函数原型：

extern “C” \_\_declspec(dllexport) VOID OnSize(

Rect rtBoard

);

参数：

rtBoard [in]：指明棋盘在SAU Game Platform主窗口客户区位置的RECT结构。

返回值：

无。

说明：

当SAU Game Platform处理主窗口的WM\_SIZE消息或进行棋种切换时需要调用OnSize函数，通知棋种支持模块绘制棋盘图像的位置。

1. DrawBoard 函数

功能：

绘制棋盘。

函数原型：

extern “C” \_\_declspec(dllexport) VOID DrawBoard(

HDC hDC

);

参数：

hDC [in]：棋盘被绘制的设备环境句柄。

返回值：

无。

说明：

当SAU Game Platform处理主窗口的WM\_PAINT消息时调用函数，在OnSize函数指定的位置上绘制棋盘图像。

1. OnRun 函数

功能：

启动对弈逻辑。

函数原型：

extern “C” \_\_declspec(dllexport) VOID OnRun(

VOID

);

参数：

无。

返回值：

无。

说明：

当用户开始对弈时即调用OnRun函数通知棋种支持模块对弈开始。

1. OnLButtonDown 函数

功能：

处理用户对着法的输入。

函数原型：

extern “C” \_\_declspec(dllexport) VOID OnLButtonDown(

int x,

int y

);

参数：

x [in]：鼠标指针在SAU Game Platform主窗口客户区的横轴坐标。

y [in]：鼠标指针在SAU Game Platform主窗口客户区的纵轴坐标。

返回值：

无

说明：

当用户着法输入有效时调用OnLButtonDown函数将用户输入交付给棋种支持模块进行处理。需要注意的是，(x,y)表示的坐标为鼠标指针在窗口客户区上的坐标，将其转换为棋盘上的坐标，则需要根据OnSize函数指定的棋盘位置进行变换。

1. OkMove 函数

功能：

确认用户着法。

函数原型：

extern “C” \_\_declspec(dllexport) VOID OkMove(

char \*moveCmd

);

参数：

moveCmd [out]：指明用户着法的“move”命令串。

返回值：

无。

说明：

当用户确认着法后，调用OkMove函数得到指明用户输入着法的“move”命令串，即产生行棋事件。

1. CancelMove 函数

功能：

取消用户着法。

函数原型：

extern “C” \_\_declspec(dllexport) VOID CancelMove (

VOID

);

参数：

无。

返回值：

无。

说明：

当用户取消着法后，调用CancelMove函数清除用户输入的着法，同时恢复棋盘图像的显示状态。

1. ProcessMove 函数

功能：

处理行棋事件。

函数原型：

extern “C” \_\_declspec(dllexport) INT ProcessMove(

char \*moveCmd,

char \*curCmd,

char \*denCmd

);

参数：

moveCmd [in]：携带行棋事件的命令串的缓存区指针。

curCmd [out]：通知当前行棋棋手裁判行为的命令串的缓存区指针。

denCmd [out]：通知对方棋手裁判行为的命令串的缓存区指针。

返回值：

指明所处理行棋事件的状态码，-1表示非法着法，2表示致胜着法，其它值表示普通着法。

说明：

当对弈逻辑控制模块获取到棋手发出的行棋事件后，即调用ProcessMove函数对行棋事件进行处理，然后得到裁判行为。

1. GetCurPlayer 函数

功能：

获取当前行棋棋手。

函数原型：

extern “C” \_\_declspec(dllexport) INT GetCurPlayer(

VOID

);

参数：

无。

返回值：

指明当前行棋棋手的数值，0表示黑方棋手，1表示白方棋手。

说明：

无。

### 实现棋种支持模块的说明

SAU Game Platform并不关注棋种支持模块的内部细节是如何实现的，只要掉用CheckModule函数时，Info参数被字符串“SAU\_GamePlatform chessType”填充即认为棋种支持模块是符合规定的。另外，因为具有良好的接口定义和自由性隔离，棋种支持模块可以由任何人进行开发并接入到SAU Game Platform中。

实现棋种支持模块时需要注意：

1. 模块内部应自行维护基本对弈逻辑，并且SAU Game Platform对于该逻辑也具有依赖性。
2. 模块只会接收到命令串形式的行棋事件，且该事件中并不指明着法属于哪个棋手，对此的辨认完全依赖模块内部维护的基本对弈逻辑。
3. SAU Game Platform仅仅指明棋盘的位置，棋盘上的内容的安排完全有棋种支持模块决定。
4. 用户输入的着法的坐标系与棋盘所在的坐标系相同，模块对用户输入着法的解析完全有模块自行定义。
5. 因为用户输入着法的确认取消机制，所以模块内部应维护用户输入着法的暂存，且该着法也将映射到棋盘图像上。

## 底层驱动——博弈引擎控制模块

博弈引擎控制模块是控制博弈引擎程序的驱动机构，如果没有博弈引擎控制模块，SAU Game Platform就不能与博弈引擎程序达成通信，平台也就失去了其存在的意义。

### 双缓冲技术获取引擎发送命令串

SAU Game Platform采用ReadFile函数读取管道中的引擎写入数据，若管道中保存的数据长度大于函数读取的数据长度，则可能造成命令串截断。为了解决这一问题，SAU Game Platform采用双缓冲技术，将本次读取的数据与上次处理后遗留的数据进行拼接，再从拼接后的数据中取得完整的命令串。

## 网络通信控制协议