# Server Components

Originally Albot.Online was created using the [“Barebones MSF Library”,](https://github.com/alvyxaz/barebones-masterserver/wiki) but has since been altered in several ways. The main concept of the Barebones workflow with masterServer, Spawners and game server(rooms) still remains. It could therefore be quite insightful to gloss over the original API.

Since Albot.Online has in several ways made modifications to the library, it’s important to note that we can not update our Barebones library.

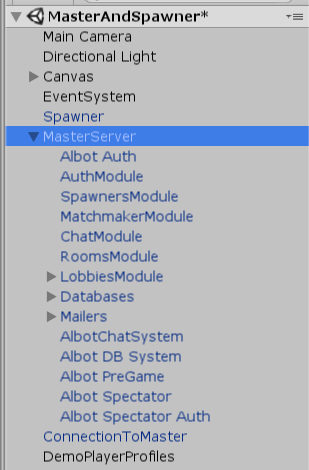
The server side of Albot.Online can mainly be split up into five big sections.

1. MasterServer
   1. Database
2. Spawner
3. GameServer
4. DLL Game logic

# MasterServer

## Description

The MasterServer is the main connection point of all other components in the Albot.Online platform. In broad terms the workload performed by the MasterServer can be summed up by the following main sections:

* Allow players to connect and handle their authentication/accounts.
* Keep track of player states. In what game that player is currently in, or if that player is in the lobby.
* Manage the player database.
* Lobby chat.
* Manage active pre-games.
* Manage active spawners.
* Keep track of running game servers.
* Connect players to the game servers.
* Distribute the game logs to spectators.

All of the mentioned logic above is divided up into several modules. For example there exists “***RoomsModule***” that is in charge of tracking running game servers and ***“Albot Auth Module***” that manages the login phase of players.  
Every module must reside as an unique object under the **“*MasterServer*”** object found in the “***MasterAndSpawne***r” scene.

## 

# Spawner

## Description

Spawne的唯一目的是生成新的游戏服务器实例。 当要创建新游戏时，MasterServer 将找到一个不繁忙的生成器并请求创建游戏。 然后由生成器创建游戏服务器，并向 MasterServer 提供对新游戏服务器的引用

To spawner logic resides in the same executable as the MasterServer, but can be turned into an spawner by being passed the argument *“****-msfStartSpawner****”* at startup.

***./MasterAndSpawner.x86\_64 -msfMasterPort 5000 -msfStartSpawner -logfile***

# GameServer

## Description

每次在 Albot.Online 服务器上启动游戏时，都会生成一个专用于该游戏的新进程。 这个过程就是所谓的GameServer。 当游戏服务器启动时，它将接收有关要玩什么类型的游戏以及期望加入哪些玩家的参数.

## Game protocol

Every game has a unique game state that has to be described by some kind of protocol. On top of this there exists two parts to every Albot.Online protocol.

* ***Internal Server Protocol***  
  The internal server protocol (*ISP*) is the protocol between the GameServer and the players Albot.Online clients. An ISP decides upon the internal game representation and can therefore be optimized on factors such as datasize. Which can become important when dealing with realtime games that send several updates every second.
* ***Bot protocol***The bot protocol is the actual interface that we display to the developers and their bots. Here we are forced to rely on a format that is easy to parse and extract data from. Sofar we have used plain text for the more simple games, and relied on Json formatting for games containing more information. See [Snake Protocol](https://www.albot.online/snake/) for an example of Json formatted protocol.

## Game logic

Naturally every game will have its own rules and logic. Therefore when an instance of the GameServer is started it will choose what game logic to use, based on the arguments passed to it by the MasterServer.

Game logic used in Albot.Online can be divided into one of two section

* ***DLL based logic***此方法利用可以创建并导入到 Unity 中的托管 DLL 文件。 这允许我们在主 Albot.Online 项目之外创建游戏逻辑，然后让 GameServer 动态导入它。 这是首选，因为它将 Albot.Online 分发到几个更容易维护的模块中。

此方法的一大缺点是我们在创建游戏逻辑时被迫使用纯 C# 代码。 这对于大多数棋盘游戏等简单游戏来说效果很好，但当游戏包含碰撞和物理等内容时，这会变得越来越困难.

* ***Native Unity based logic.***与 DLL 方法相比，直接在 Albot.Online Unity 项目中构建游戏逻辑使我们能够访问游戏引擎中的所有功能。 使用此方法时，新的 Unity 游戏场景专用于游戏，因此逻辑的实现就像常规 Unity 游戏一样。 最大也是唯一的缺点是这降低了项目的模块化程度.

For a more detailed and technical explanation of how to create new Albot.Online games see:  
“Albot.Online Game Creation”

Turnbased & Realtime  
所有使用服务器的游戏都可以分为实时游戏或回合制游戏。 服务器处理这两种不同类型游戏的最大差异是，在实时游戏期间，服务器将不断向所有连接的玩家发送更新，同时监听所有玩家的输入。

相比之下，回合制游戏将忽略除当前活跃玩家之外的所有输入。 当它接收到来自当前玩家的输入时，它将向所有玩家广播更新，其中包含有关所采取行动的信息.

## Selfplay

大多数回合制游戏的公式通常可以轻松创建一个机器人，该机器人能够仅根据发送给它的当前信息做出决定。 这使得我们通常可以创建一个可以轻松转换以匹配对手视图的游戏协议。 使我们能够简单地重新解析当前的游戏状态，就好像玩家也在扮演对手一样。 允许机器人轻松地与自己对战，而无需开发人员的任何麻烦. For an example see the [Connect4 protocol.](https://www.albot.online/connect4-1/)

然而，实时游戏的本质使得自我游戏变得更加乏味，因为机器人能够在处理多个玩家的同时仍然做出快速决策。 通常实时游戏需要机器人保留游戏的内部表示。 开发人员创建可以处理自我游戏的实时机器人变得更加复杂。 因此我们没有启用这个选项.

Typical Game Flow

## Description

This section describes the backend server-process that happens when two users create, join and then play a game against each other, using the Albot.Online gameserver.

## Prerequisites

To make good use of this section, make sure that you have a good understanding of the [components that make up the Albot.Online server.](#_s444q1ctvnjv)

### Pre-Game phase:

当用户决定创建一个新的在线游戏时，masterServer将实例化一个预游戏。 所有预赛都显示在大厅中，如果未满则允许其他玩家加入。 当赛前所有玩家都准备好时，主机玩家可以选择开始游戏。

### Game setup phase:

Once the pre-game is given the start signal there are a series of initializing events that must occur before the game can begin.

1. A free spawner is found.
2. Found spawner will create a new game server instance. Whilst doing so it passes arguments regarding what type of game is to be played, the names of the pre-game players and their corresponding order.
3. The game server initializes.
4. MasterServer invites pre-game players to the newly created game server.
5. Players connect to the game server.
6. Once all the predefined players has joined the game server, the game will start.