

# 项目《天眼 (Sky EYE) 》草案

## 1. 相应准备

技术选型:

Git

Python version 3.10.6

Tornado version 6.2

PyQt5 version 5.15.9

数据库待定, 预设是 mongodb

通信协议待定

tips: 开发环境可以用 VS Code, 参考插件可以是 autoDocstring、Bracket Pair Color DW、Git Graph、Github Copilot、PyLance、LiveCode for python、Python Indent 等。

故事背景来源: 起点中文网山有十四著《青天有眼》

## 2. 项目框架

MUD\_GAME

|-- Server

    |-- config 配置文件夹, 暂时没啥用

    |-- controllers 服务端 world state 维护对象, 如 actor 等

    |-- routes 路由, 网络配置

    |-- tests 测试集, 暂时没啥用

    |-- utils 工具集

    |-- scripts 脚本

    |-- mud\_server\_app.py

|-- Client

    |-- core 核心代码

        |-- editor 编辑器

            |-- mainwindow 主窗口

            |-- api 基类集, 存放 plugin、widget 的基类

                |-- plugins

                |-- widgets

```
|-- plugins 插件集，继承 api
|-- utils 工具集
|-- routes 路由，网络配置
|-- resources 文本、图片、音频资源
|-- scripts 脚本
|-- mud_client_app.py
|-- Docs 开发日志、策划草案等存放
|-- README.md
```

tips:

1. 有些文件夹暂时是空的，所以 git 上面没有，git 只能保存有东西的路径。
2. utils 工具集主要存放一些在该端可以共用的东西，即比如在服务端中某些全局变量，就可以放在 utils 中。

### 3. 开发进程

预期 2023/09/23 推出 v1.0

具体时间安排：

2023.05 之前，尽可能完成基础框架搭建与数据库、通信协议选定，包括网络通信、插件开发、tick 机制、日志监控、热更新配置等。

2023.09 之前，完成游戏内容填充，于此同时可以选择推进一定的免费宣传。

2023.09 之后，进行服务器压力测试等测试内容，并且完善通信安全设计与编码，进行最后的整体 debug。

tips:

1. 开发某一模块时，就应该完成该模块的单元测试，至少保证模块可用，再接入主项目中，再次调试且保证接入可用。
2. 注意线程、协程资源分配，目前 PYQT 的 QApplication 必须执行在主进程、主线程，所以 tornado 客户端是以线程形式调用；
3. 游戏内容将依据《青天有眼》的大纲、世界背景进行设计，主要分为任务系统（剧情任务、活动任务等）、战斗系统、技能树、物品系统、合成系统（装备、消耗品制作）等。但不是很急。
4. 开发分支规范：vX.X 分支将作为合并分支，自己的提交内容另外新建一个固定的分支如 test\_lyq 或者以模块名命名如 journal，审核、调试无误后，再合并到当前版本分支当中。合并由胡俊康负责。
5. 开发内容应该实时记录在 development journal.md 文件中，以便将来复盘。

## 4. 注释规范

以 connect\_server\_app.py 部分为例

```
MUD_Game > Server > routes > connect_server_app.py > Game_Server > add_client
1  # -*- coding: utf-8 -*- #
2
3  # -----
4  # Topic: connect server end with tornado
5  # Author: k14
6  # Created: 2023.04.04
7  # Description: holding the control of tornado server and game server
8  #             test port: 8080
9  # History:
10 #   <author>   <version>   <time>       <desc>
11 #   k14        v0.1        2023/04/04    basic build success
12 # -----
13
14 import tornado.web
15 import tornado.websocket
16 import tornado.ioloop
17
18 import base64
19 import hashlib
20 import logging
21 import datetime
22
23
24 class Game_Server:
25     """_summary_
26
27     it tackles the core logics about game server.
28
29     Attributes:
30     |   clients_list: all client connected the game server.
31     """
32     def __init__(self):
33         self.clients_list = set()
34
35     def add_client(self, client):
36         """_summary_
37
38         Add the client into the game server.
39
40         Attributes:
41         |   client: Tornado_Client_App
42         """
43         self.clients_list.add(client)
44
45     def remove_client(self, client):
46         """_summary_
47
48         Remove the client into the game server.
49
50         Attributes:
51         |   client: Tornado_Client_App
52         """
53         self.clients_list.remove(client)
54
```

1. 每份代码文件头都应该注释该代码文件的具体功能；
2. 类、函数都应采取如图模板进行注释；
3. 考虑到项目后期可能会将早前版本开源，所以最好是用英文进行简单注释；
4. import 规范：以 mud\_client\_app.py 为例

```
# -*- coding: utf-8 -*- #

# -----
# Topic: client main app
# Author: k14
# Created: 2023.04.09
# Description: the main client including interface and communication.
# History:
#   <author>    <version>    <time>        <desc>
#   k14         v0.1         2023/04/09    build the basic
# -----

from core.editor.editor_app import Editor_Main_Window
from core.routes.connect_client_app import Tornado_Client_App, Tornado_Thread

from PyQt5.QtWidgets import QApplication

import sys

class SE_Client_App:
    """ summary
```

优先 import 代码文件，即项目中的代码文件；然后 import 第三方多模块，如 PyQt5、Tornado 等；最后 import 系统库及一些琐碎的第三方库，如 sys、logging 等；

5. 注意空格行，注释头与 import 间隔 2 行、import 与类间隔 2 行、类与类间隔 2 行、函数与函数应该空格 1 行、类与主程应该空格 2 行；

6. 文件夹名以小写命名，比如 controllhelpers，不要空格或者下划线；代码文件名以小写命名，不要空格或者下划线；类名以驼峰命名，如 TornadoServerApp，不要空格或者下划线；函数名以小写加下划线形式，如 init\_ui，不要大写也不要空格。

7. 变量名规范：如果是类与类之间交互的变量，即公开变量，应该以小写驼峰形式命名，其中信号类参数应命名为 signal\_xxx，如 signal\_close；如果是只有自己类用到的属性，或者对其他类来说该属性应该只读，即私有属性，命名应该采用\_\_attribute\_name\_\_形式，前后各四根下划线，这将为了避免与继承类原私有属性发生冲突，且利用装饰器进行只读处理，例如：

## 5. 装饰器:

类成员装饰器:

```
python

# @property: 广泛运用于类当中，将函数转换为只读方法，可以用来设置python类中的私有成员；（getter）
# @xxx.setter: 用于修饰xxx（属性名），设置对应属性的数据。

@property
def name(self):
    return self.__name__

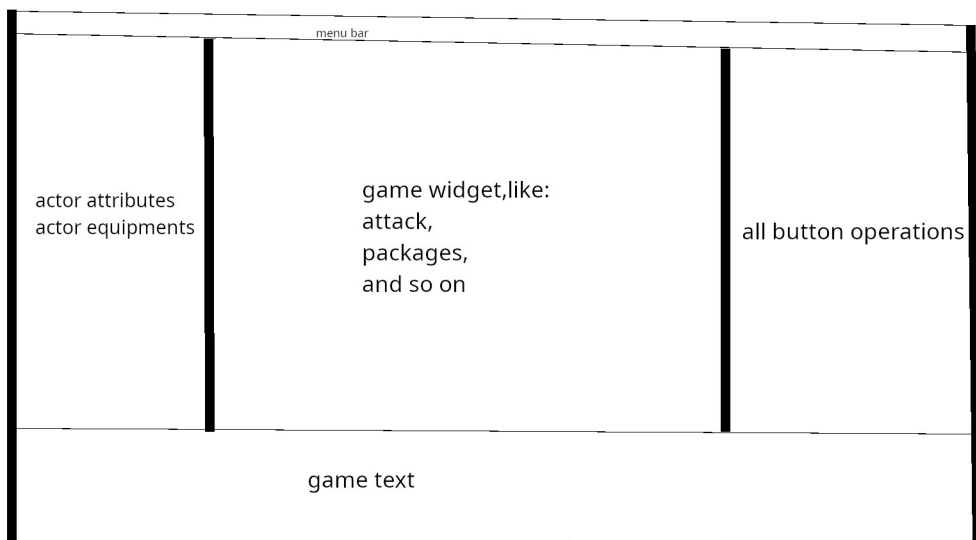
@name.setter
def name(self, data):
    self.__name__ = data

# @classmethod: 所修饰的函数将不需要实例化，且无需带self函数。但定义时，第一个参数必须是表示自身类的 cls 参数；
# @staticmethod: 声明静态函数，可不带任何参数（包括cls、self）；可不实例化类即调用；
```

## 5. 游戏内容

游戏内容包括但不限于界面 UI、游戏系统设计、资源（文本、图片、音频）等，将会在开发全进程中逐渐填充。

### （1）界面 UI 预期效果



鉴于我们已经使用了 QDockWidget，完成图片所述布局将会非常简单。但应该注意每个 widget 的细节设计，例如：

a. 一些重要窗口，如 actor attributes、actor equipment（这是两个窗口）误关闭后，应该能在 menu bar 中重新开启；

b. 由于 QDockWidget 已经实现了拖拽成单独窗口、回拽到主窗口的逻辑，我们需要保证回拽窗口应该出现在我们希望的位置，也就是它原本在主窗口的位置；

c. 刚刚打开游戏客户端时，**game widgets** 应该处于 **hide** 状态的，它们的 **show** 与 **hide** 将取决于 **button operation**，以及其他执行逻辑；

d. **game text** 主要存放游戏过程中可能产生的一切文本，包括但不限于游戏对话、战斗描述、战斗获得等；

e. **game text** 的位置可能与 **all button operations** 的位置作出调换，具体看实现效果，修改这个只需要修改注册插件时的位置，非常简单。

f. **menu bar** 不要用 **QDockWidget** 来辅助构建，它应该直属于 **main window**。