写给小白

大佬们可以跳过该部分.

- 请严格按照教程操作, 不要想当然.
- 如果实在不知道怎么安装或者安装后怎么都用不了, 可以进用户群(qq)问: 568500514

前期准备

学习简单使用命令行, 如果你不愿意花时间, 安装好之后可以使用 GUI 版本, GUI 版本相比于纯后端版本功能较少, 以后会逐步添加.

安装 雷电模拟器, 战舰少女R.

雷电模拟器建议安装雷电模拟器9.

安装 Python 注意, Python 版本要求 3.7 <= x <= 3.9, 我们推荐你安装 Python3.7.9.

写给小白: 如果你之前还安装过 Python, 请卸载掉之前安装的版本, 进行上述操作时关闭计算机上的一切第三方杀毒软件.

将模拟器设置为 `>=1280x720` 的 `16:9` 分辨率(推荐使用 1280x720 分辨率).

.

安装本项目(AutoWSGR)

两种方式二选一.

PyPI 安装

AutoWSGR 目前已支持通过 PyPI 进行部署, 在安装好 Python 后, 打开命令提示符(cmd), 输入以下命令后回车.

```
pip install -U AutoWSGR
```

Github 安装

你也可以通过以下命令从 GitHub 安装最新版 (推荐, 本项目尚处于开发早期, 这里 bug 修复更及时):

```
pip install -U git+https://github.com/huan-yp/Auto-WSGR.git@main
```

检查是否安装成功

`Win+r` 打开 "运行", 输入 `python` 后回车, 在打开的黑框框里输入以下代码:

```
import AutoWSGR
print(AutoWSGR.__version__)
```

出现以下输出即为安装成功(主要是数字 `0.0.9`, 这个数字可能会随着本项目更新变化.)

```
Python 3.7.16 (default, Jan 17 2023, 16:06:28) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import AutoWSGR
>>> print(AutoWSGR.__version__)
0.0.9
>>>
```

配置用户文件

这一份简单的启动代码:

```
from AutoWSGR.scripts.main import start_script
timer = start_script()
```

这份代码启动了整个程序并获取了一个控制器 `timer`, `start_script()` 可以有参数, 代表用户设置的路径, 例如:

```
from AutoWSGR.scripts.main import start_script
timer = start_script("user_settings.yaml")
```

如果不指定这个参数, 程序将会使用默认的用户配置文件运行, 默认文件位于本仓库 `/src/AutoWSGR/data/default_settings.yaml`.

如果某些参数在你的配置文件中没有指定, 则会使用默认配置文件中的参数.

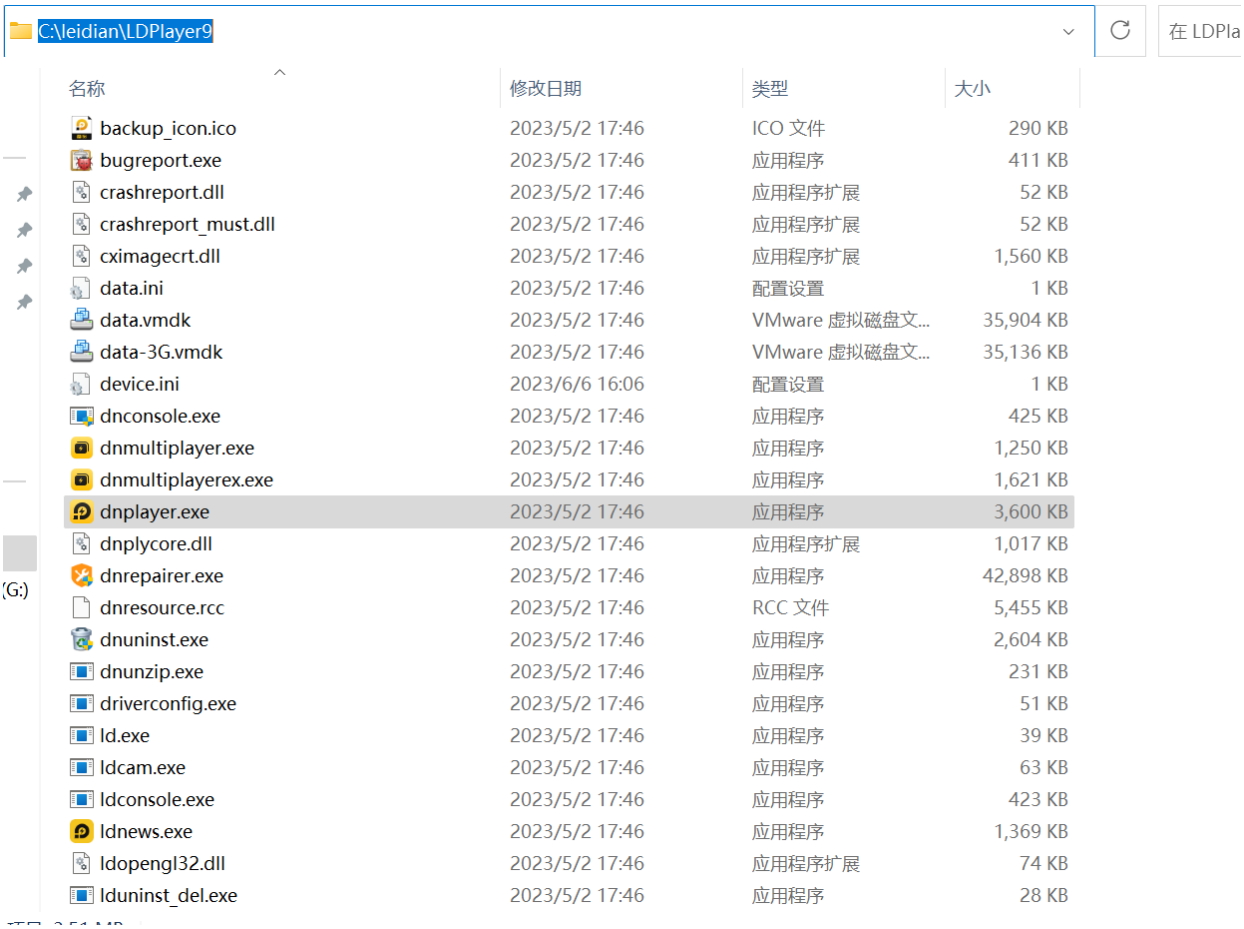
配置文件模板如下:

```yaml
# ========== 游戏设置 ==========
emulator:
  type: "雷电" # 模拟器类型
  start_cmd: "C:/leidian/LDPlayer9/dnplayer.exe" # 你的雷电模拟器启动路径
  config_file: ""

LOG_PATH: "log" # 日志记录路径
DELAY: 2 # 延迟倍率

account: # WSGR 用户名
password: # 密码

# ========== Logger设置 ==========
DEBUG: True
SHOW_MAP_NODE: False
SHOW_ANDROID_INPUT: True
SHOW_ENEMY_RULES: True
SHOW_FIGHT_STAGE: True
SHOW_CHAPTER_INFO: True
SHOW_MATCH_FIGHT_STAGE: True
SHOW_DECISIVE_BATTLE_INFO: True
SHOW_OCR_INFO: True
log_level: DEBUG # 调试模式 log_level 应该设置为 DEBUG

# ========== 日常挂机策略设置 ==========
daily_automation:
  auto_expedition: True # 自动重复远征
  auto_gain_bonus: True # 当有任务完成时自动点击
  auto_bath_repair: True # 空闲时自动澡堂修理

  dock_full_destroy: True # 船坞已满时自动清空，若设置为 False 则船坞已满后终止所有常规出征任务

  auto_battle: True # 自动打完每日战役次数
  battle_type: "困难驱逐" # 战役类型

  auto_normal_fight: True # 按自定义任务进行常规战
  normal_fight_tasks: # 自动出征任务用列表 [plan名, 舰队号, 目标成功次数] 表示，按顺序从上往下执行.
    - [8-5AI, 2, 0]

# 数据信息设置
SHIP_NAME_PATH: user_settings/ship_name.yaml # 舰船名称文件位置
PLAN_ROOT: "" # 策略文件根目录
```

下面会提几个重要参数, 没有提到的参数一般都可以使用默认值.

启动路径(start_cmd)

如果你每次使用时都手动先打开模拟器, 可以不用配置模拟器启动路径, 否则, 找到雷电模拟器的安装目录, 复制过来, 并在后面加上程序名. 例如我的雷电模拟器安装路径在这里:



因此我需要填写 `C:/leidian/LDPlayer9/dnplayer.exe`.

账号密码(account)

一般来说可以不填写, 程序会默认已经完成登录并进行对应操作, 如果提供了账号密码, 程序会首先尝试退出当前账号, 退出后再使用账号密码登录.

日志信息(log_level)

为了方便检查程序的 bug, 默认启用最详细的日志信息, 如果你觉得输出的信息太多了可以改为 `INFO` 或者 `ERROR`.

日常挂机策略

battle_type

一共十个选项:

1. 简单驱逐
2. 简单巡洋
3. 简单战列

4. 简单航母
5. 简单潜艇
6. 困难驱逐
7. 困难巡洋

   ...

normal_fight_tasks

下一部分会介绍 plan.

配置策略文件(plan)

我们通过一个文件来描述一场战斗的策略, 这个文件被称作 plan, 它的后缀名是 `.yaml`, 查看文件的后缀名.

注意, .yaml 格式对缩进格式有要求, 你可以将同样的缩进条目认为是处在同一个文件夹下.

所有参数大小写敏感.

全局策略文件

为了简化配置, 我们提供了一个全局策略文件, 它指定了一些默认参数, 如果在具体的策略文件中没有指定相关参数, 将会使用全局参数, 以下是我们的全局策略文件, 它位于本仓库 `/src/AutoWSGR/data/plans/default.yaml`. 内容如下:

以下面的例子简单了解下 yaml 的缩进, 你可以认为 normal_fight_defaults 是一个文件夹, 包含了 chapter, map 等参数, battle_defaults 又是另一个文件夹, 包含了其它参数, 当然, '文件夹' 内可以包含 '文件夹'.

```yaml
normal_fight_defaults: # 常规战斗的默认参数(包括活动)
  chapter: 1 # 章节
  map: 1 # 地图
  repair_mode: 2 # 1:中破就修, 2:只修大破, 也可以用列表指定 6 个位置不同修理方案
  selected_nodes: [] # 选择要打的节点, 白名单模式, 一旦到达不要的节点就撤退或者 SL
  fight_condition: 1 # 战况选择
  fleet_id: 1 # 默认舰队
  fleet: null # 舰队成员 (按名字区分), 为 null 则保持为当前选择的舰队不改变
  # fleet_id 为 1 时不支持指定舰队成员
  # fleet: ["", "吹雪", "明斯克", "胡德", "赤诚", "",]
  # 填 "" 或者不填则代表该位置无舰船
  # 0 号位(第一个)填 "" 占位
  # 上例的舰队最终为: 吹雪 明斯克 胡德 赤诚 5 号位无舰船 6 号位无舰船, 旗舰为吹雪
  # 现有策略填写时用的作者的船舱名, 记得修改
  # 如果舰船名为纯数字, 必须加双引号, 如果不是纯数字可以省略双引号.
  # 请保证舰队配置符合规定 (不能出现相同舰船, 舰船占用前若干个位置)

battle_defaults: # 战役的默认参数
  map: 1 # 简单 1-5 困难 6-10
  repair_mode: 1 # 1:中破就修, 2:只修大破, 也可以用列表指定 6 个位置不同修理方案 [1, 1, 1, 1, 1, 1]

node_defaults: #
  # 演习特有的参数
```

```
  fleet_id: 1 # 队伍编号,可以在自己的方案中设置,机器人和玩家独立
  max_refresh_times: 2 # 默认最大刷新次数,可以分别对玩家和机器人设置
  # 索敌成功阶段
  detour: False # 是否迂回, 如果该节点不支持迂回或者索敌失败将忽略该参数, 否则如果选择迂回则进行迂回操作
  enemy_rules: [] # 每条按照 ["(类型 符号 数量) and/or ()", "操作"] 的字符串形式给出，从上到下执行第一条符合的命令。
                  # 类型：舰船类型；符号：[>=、<=、>、<、==、!=]；数量：数字；操作：[retreat、detour、阵型数字(1-5)]
                  # 如果在这里做了阵型选择决定，则优先级将高过下面的阵型选择。例子：["(SS >= 2)", "5"]
                  # and 的优先级高于 or
  # 阵型选择阶段
  SL_when_spot_enemy_fails: False # 索敌失败时是否 SL
  SL_when_detour_fails: True # 是否迂回失败后退出游戏
  SL_when_enter_fight: False # 进入战斗后是否退出游戏
  formation: 2 # 正常情况阵型选择, 1-5
  formation_when_spot_enemy_fails: False # 启用时改为数字, 索敌失败时阵型选择, 1-5
  # 夜战选择阶段
  night: False # 是否进入夜战
  # 前进选择阶段
  proceed: False # 结束后是否继续
  proceed_stop: [2, 2, 2, 2, 2, 2] # 根据我方血量状态选择是否继续前进, 一旦对应破损程度达到或超过该值则返回
                                   # [1,2,2,2,2,-1]: 旗舰中破回家, 2-5 号位大破回家,忽略 6 号位的状态

  # 特殊情况
  supply_ship_mode: 0 # 0：无额外操作  1：有补给舰则战斗, 否则撤退

exercise_defaults:
  chapter: 'exercise' # 不要改动
  selected_nodes: [player, robot] # 不要改动
  discard: False # 不要改动
  exercise_times: 4 # 最大玩家演习次数
  robot: True # 是否打机器人
```

下面会对上面的重要部分做出一些补充.

normal_fight_defaults

repair_mode

指定修理模式, 会在开始战斗前进行修理操作, 如果破损程度超过阈值则进行修理, 例如指定为 1 时, 对应位置舰船中破或者大破后修理, 否则只修大破, 如果以列表形式指定, 长度必须为 6, 值必须为 1~2, 否则运行时可能会报错. 如果以数字的形式指定, 必须为 1 或 2.

例如指定为 [1,1,2,2,2,2], 则 1,2 号位的舰船中破或者大破后就会修理, 3-6 号位的舰船中破不修理, 大破修理.

我们暂时不提供大破出击的支持.

selected_nodes

指定哪些节点要经过, 如果到达了没有指定的节点, 会尝试撤退, 如果索敌失败, 则采用 SL 的方式撤退, 注意, 如果索敌失败并且没有进入阵型选择(少于 4 船)直接进入了战斗, 可能会出现不可预料的行为, 大概率会在进入战斗后 SL, 小概率会报错.

例如指定 ["A", "B", "D"], 则到达 C, E, 等其它点时会直接撤退, 到达 A, B, D 后则会进入下一步的决策.

fight_condition

选择的战况, 编号为 1-5, 几何分布如下, 暂时不支持对于同一张地图的两个战况进行不同的选择:

```
1     3
   2
4     5
```

battle_defaults:map

该参数指定默认的战役地图, 一般用的比较少, 在自行编写战役策略时可能有用. 推荐使用我们提供的 5 种战役的全部完整策略.

node_defaults

node_defaults 是为每个地图中的节点 (A, B, C 这些), 指定的默认策略, 如果对于某个 selected_nodes 中存在的节点, 你没有单独为它写策略, 则会使用这个默认策略, 在单个地图的策略文件中, 你会知道如何为一个节点单独编写策略.

fleet_id && max_refresh_times

这两个参数是为演习准备的, 但是我们提供了演习的完整策略, 推荐使用该策略.

整个 5 次演习的过程逻辑上被视为了一张地图, 每个节点可以选择不同的舰队参数.

enemy_rules && formation && formation_when_spot_enemy_fails

formation 和 enemy_rules 大致确定了阵型选择策略, 阵型选择按以下顺序进行

如果索敌失败并且没有SL, 如果设置了 formation_when_spot_enemy_fails, 则选择 formation_when_spot_enemy_fails 指定的阵型, 如果没有设置则选择 formation 指定的阵型.

如果索敌成功, 将优先考虑 enemy_rules 指定的阵型, enemy_rules 通过逻辑条件指定阵型.

每条按照 ["(类型 符号 数量) and/or ()", "操作"] 的字符串形式给出, 从上到下执行第一条符合的命令。
类型: 舰船类型, 符号: [>=、<=、>、<、==、!=、+、-、*、/], 数量: 数字, 操作: [retreat、detour、阵型数字(1-5)]

舰船类型如下:

```
轻巡:CL, 重巡:CA, 雷巡:CLT, 航巡:CAV, 大巡:CBG, 导巡:KP, 防巡:CG,
航母:CV, 轻母:CVL, 装母:AV, 战列:BB, 航战:BBV, 战巡:BC, 重炮:BM,
驱逐:DD, 炮潜:SC, 潜艇:SS, 常规补给:NAP, 特殊补给: SAP,
导驱:ASDG, 防驱:AADG, 导战:BG,
```

NAP:常规补给舰, 除了掉战利品的之外的所有补给舰.
SAP:特殊补给舰, 战利品期掉战利品的蓝色补给舰.

enemy_rules 应该是一个列表, 以下是一个示例:

```
enemy_rules:
  -
    - SS > 0 or SC > 0
    - 5
  -
    - BB + BC + BG > 0
    - retreat
```

```
enemy_rules: [[SS > 0 or SC > 0, 5], [BB + BC + BG > 0]]
```

这两个示例描述的逻辑是同一个, 先看是否有炮潜或者潜艇, 如果有就采用单横阵, 如果没有, 则检查战列, 战巡, 导战的总数是否大于 0, 如果大于 0 就撤退. **注意, 如果有潜艇并且有导战, 那么由于潜艇的逻辑先满足条件, 因此会用单横阵进入战斗而不是撤退, 如果希望有战列等船时先撤退, 应该调换两个逻辑的顺序.**

这些运算存在优先级关系, 具体的, 乘除法 > 加减法 > 比较大小 > 逻辑与操作 > 逻辑或操作, 你可以使用括号来改变运算顺序, 如果你需要一些比较奇特的运算优先级或者不清楚运算优先顺序, 我们建议你使用括号而不是盲猜一门你不了解的编程语言的运算顺序.

如果两个条件都不满足, 则使用 formation 指定的阵型.

你可以查看 我们提供的一些基本策略 以获得更多的实际示例.

yaml 的列表有两种写法, 以上两个示例本质相同写法不同. 注意第一种写法的缩进问题.
我想虽然支持但是应该没有人会用到乘法和除法吧.

proceed_stop && proceed

这两个参数决定了战斗结算之后是否选择回港, 如果 proceed 指定为 True, 则选择继续前进, 如果 proceed 指定为 False, 则选择回港, 如果出现旗舰大破或者燃油耗尽, 会自动回港.

proceed_stop 参数则是一个列表, 它决定是否应该在某些舰船破损程度达到一个限度时回港.

当指定为 [1, 2, 2, 2, 2, -1] 时: 旗舰中破回家, 2-5 号位大破回家, 忽略 6 号位的状态(6 号位大破可以继续前进).

当然, 你也可以这么写:

```
proceed_stop:
  - 1
  - 2
  - 2
  - 2
  - 2
  - -1
```

supply_ship_mode

很显然这个参数可以通过指定 enemy_rules 实现.

exercise_defaults

这些参数指定了演习策略文件的默认参数, 我们并不建议你修改它.

常规战斗策略

常规战斗指常规图的战斗以及除 "立体强袭" 之外所有活动的战斗, 每一个常规战斗任务都应该指定一份战斗策略, 一份常规战斗策略的模板如下:

```yaml
# MapLevel
chapter: 7 # 章节
map: 4 # 地图
selected_nodes: [B, E, D, G, L, K, M, H] # 选择要打的节点，白名单模式，一旦到达不要的节点就SL
repair_mode: 1
fleet_id: 2
fleet: ["", "U-1405", "U-1206", "狼群47", "射水鱼", "U-96", "肥鱼"]

node_defaults:
  formation: 2 # 正常情况阵型选择，1-5
  # 夜战选择阶段
  night: False # 是否夜战
  enemy_rules:
    - [DD + CL <= 1 and ((CVL == 1 and CV == 0) or CVL == 0), 4]
  # 前进选择阶段
  proceed: True # 结束后是否继续
  proceed_stop: [2, 2, 2, 2, 2, 2] # 根据我方血量状态选择是否继续前进,一旦对应破损程度达到或超过该值则返回

node_args:
  B:
    enemy_rules:
      - [CL + DD >= 3, retreat]
  M:
    night: True
    formation: 4
    enemy_rules: []
```

一般来说, 你需要指定 chapter, map 等参数, 如果你不指定, 则会使用全局策略文件中的参数.

大部分参数都在上面介绍过了, 下面提几个重点

node_defaults:

如果这个参数(文件夹)存在, 将会覆盖全局策略文件中的参数, 你当然可以根本不写这个参数, 甚至空文件也是一份合法的单点战斗策略(所有参数都使用全局策略文件的参数).

这个参数有若干个子参数, 例如 proceed 等, 一个子参数如果不存在则会使用全局策略文件中的对应子参数, 否则使用这里指定的参数.

node_args:

这个参数用于单独指定每个节点的战斗策略, 它可以有若干个子参数(类比子文件夹), 每个子参数又可以包含多个参数.

如果一个节点在 selected_nodes 中但没有被指定, 那么它将会使用 node_defaults 中的参数, 如果本个文件的 node_defaults 也没有指定这个参数, 则使用全局策略文件中对应的参数.

如果一个节点在 node_args 中被指定了一部分参数, 没有指定的参数会使用本文件中 node_defaults 中的参数, 如果本文件未指定则使用全局参数.

本文件解释

这是 7-4 周常的模板, 用第二舰队, 需要打的节点已经指定了, 在 M 点采用梯形阵并且夜战, B 点如果反潜船超过或等于 3 艘就跑路, 其余节点如果反潜船可以确定的被 U1206 或者射水鱼狙击就梯形阵, 否则采用复纵阵, 道中不夜战, 只要有大破就停止前进...

fleet 参数可以指定使用哪些舰船, 第二舰队的舰船会被替换为她们, 启用这个功能需要配置舰船名称文件, 下面有说明.

战役策略

我们提供了 5 种战役的最优配置文件, 建议使用它们, 位于本仓库 `/src/AutoWSGR/data/plans/battle/`, 以下参数是困难巡洋的参数.

```yaml
# MapLevel
map: 7 # 简单1-5 困难6-10
repair_mode: [1, 1, 1, 2, 2, 2]
# NodeLevel
node_args:
  # 阵型选择阶段
  formation: 2 # 正常情况阵型选择, 1-5
  # 夜战选择阶段
  night: True # 是否夜战
```

额, 相当简单易懂吧, 由于巡洋战役是第二个, 所以困难巡洋的编号是 7, 1-3 号位是输出, 因此中破修, 其余辅助大破修.

演习策略

我们提供了一份演习配置文件, 建议使用它, 位于本仓库 `/src/AutoWSGR/data/plans/exercise/plan_1.yaml`, 以下是它的参数.

```yaml
# 功能: 演习默认模块
```

```yaml
exercise_times: 4
robot: True
node_defaults: # 如果部分参数没有指定, 将继承默认模块中的值
  night: True
node_args:
  player: # 如果部分参数未指定, 将继承 node_defaults 中的值
    formation: 4
    enemy_rules:
      - [SS >= 1, refresh]
      - [BB + BG + CBG + BC >= 2, refresh]
      - [CV + AV >= 2, refresh]
    night: True
    max_refresh_times: 2
  robot:
    formation: 1
    enemy_rules: []
```

注意到我们没有指定 fleet_id, 因为 fleet_id 应该是每一个节点的参数!(注意全局配置文件中 node_defaults 中的两个参数)

player 和 robot 是演习的两个节点, 对于所有玩家采用相同的策略, exercise_times 指定了玩家演习次数, robot 指定是否打机器人。

可以对玩家和机器人指定不同的舰队演习, 这个参数应该设置在 node_args 中对应的部分或者在 node_defaults 中指定.

max_refresh_times 指定了最多刷新对手几次, 如果刷新次数不够, 就忽略该参数.

决战策略

我们已经设计了一个接近最优的默认策略, 不需要做出改动. 决战的使用见 "决战".

策略编写指南

我们建议你复制一份全局策略文件, 并删掉不需要的部分, 修改对应参数以编写自己的策略文件, 我们提供了较多的基本策略文件供你参考, 位于本仓库 `/src/AutoWSGR/data/plans`, 也就是这里.

开始使用

写给小白: 请确保你已经**认真阅读**了开头的命令行使用教程并且了解了基本概念, 否则请使用 GUI 版本.

我们假设你已经知道了策略文件相关的内容, 以我们的策略文件目录下的 `/normal_fight/8-1A.yaml` 为例, 现在我们要执行这个策略.

用户设置文件中的 `PLAN_ROOT` 参数指定了策略库的根目录位置, 因此你应该把所有的策略都放到同一个文件夹下(可以建立子文件夹), 如果不填写 `PLAN_ROOT`, 它会被设置为一个对你来说隐藏的目录(即 `[python packages 目录]/AutoWSGR/data/plans/`), 这个目录下的文件结构和本仓库中的 `/src/AutoWSGR/data/plans` 目录完全相同.

你可以将我们的策略文件复制到另一个目录中并将 PLAN_ROOT 重新设置以使用.

下面的代码能够运行这一份策略文件:

```python
from AutoWSGR.scripts.main import start_script
from AutoWSGR.fight.normal_fight import NormalFightPlan

timer = start_script()
fight_plan = NormalFightPlan(timer, "normal_fight/8-1A.yaml") # 注意这里不是"/normal_fight/8-A.yaml", 这样是错的, 会被解析成绝对路径, 前面我们写成 "/normal_fight/8-1A.yaml" 是因为已经指定了根目录, 你的计算机的根目录是 "C:/", 而不是我们设置的 PLAN_ROOT.
fight_plan.run_for_times(100)
```

前两行代码导入了所需的模块, 第四行生成了一个控制器 `timer`, 第五行通过 `NormalFightPlan` 传入两个参数, 两个参数均为必选参数, 第一个参数为控制器, 第二个参数为策略文件的位置(以 PLAN_ROOT 参数为根目录, 注意这里不能使用相对路径).

我们的 `8-1A.yaml` 没有指定 `fleet_id` 参数, 因此会使用全局策略文件中的参数, 也就是第一舰队, 你可以修改 `8-1.yaml`, 也可以直接在代码中指定具体的舰队, `NormalFightPlan` 的第三个参数可以指定舰队的编号, 这个编号的优先级高于配置文件,

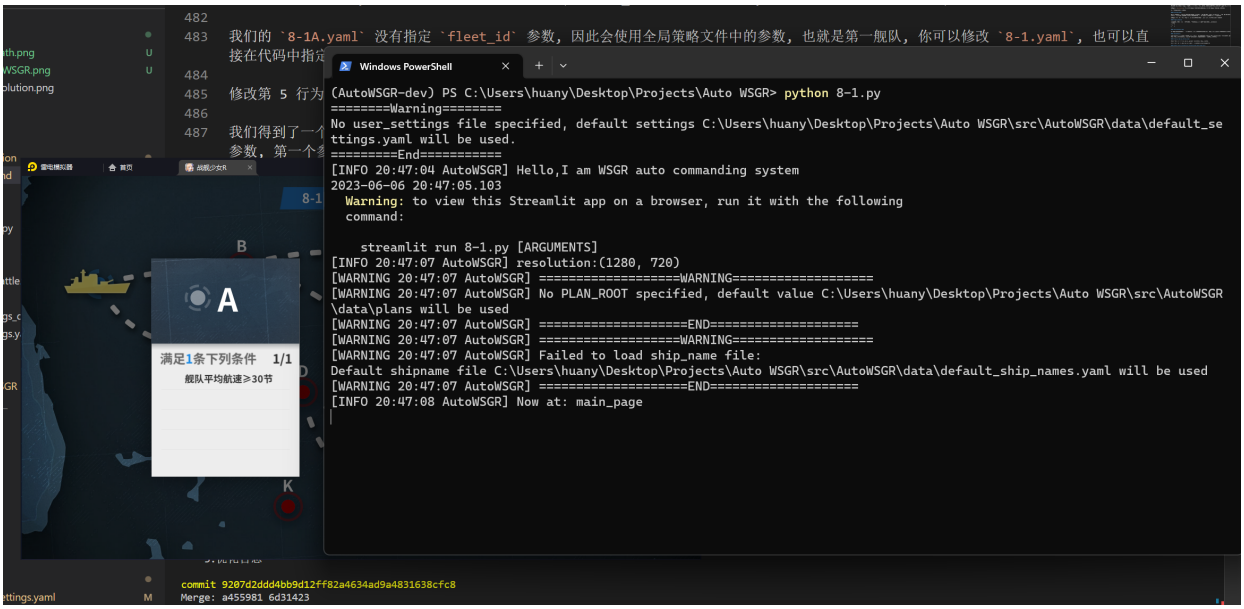
修改第 5 行为 `fight_plan = NormalFightPlan(timer, "normal_fight/8-1A.yaml", fleet_id=3)`, 即可指定使用第三舰队执行任务.

我们得到了一个配置好的任务, 然后还需要运行它, 第六行使用了 `fight_plan.run_for_times()` 运行这个任务, run_for_times() 有两个参数, 第一个参数为运行次数, 这里是 100 次, 第二个参数指定了检查远征的频率, 单位为秒, 默认值是 1800.

写给小白: 建议学习 Python 的基本语法以更好的驾驭后端, 否则请在我们提供的 example 的基础上做小修改或者使用 GUI.

安装好 AutoWSGR 后, 我们可以编写一个文件, 例如 `8-1.py` 来保存这段代码, 在我们的工作目录下, 使用命令行执行命令 `python 8-1.py` 来运行这段代码, 程序就会开始接管模拟器并进行操作了.

如图(使用时注意你的舰队编号和索敌值, 8-1A 稳定索敌线为 120):



我们的 examples 位于本仓库的 `/examples/` 下, 也就是这里.

请参考我们提供的函数原型使用.

重要接口

这里会给出一些重要的函数原型, 供有一定基础的同学参考使用.

***Plan

- AutoWSGR.fight.normal_fight.NormalFightPlan
- AutoWSGR.fight.exercise.NormalExercisePlan
- AutoWSGR.fight.battle.BattlePlan

```
class ***Plan(FightPlan):

  def __init__(self, timer: Timer, plan_path):
    """
      timer: start_script 函数提供的控制器
      plan_path: 相对于 PLAN_ROOT 参数的策略路径, 可以使用绝对路径, 例如
`C:/Users/admin/desktop/AutoWSGR/8-1A.yaml`
    """
  def run_for_times(self, times, gap=1800):
      """多次执行同一任务, 自动进行远征操作
      Args:
          times (int): 任务执行总次数
          gap (int): 强制远征检查的间隔时间
      """

  def run_for_times_condition(self, times, last_point, result='S', insist_time=900):
    """有战果要求的多次运行
    :使用前务必检查参数是否有误, 防止死循环
    :Args
        times : 次数
        last_point: 最后一个点
        result: 战果要求 (>=)
        insist_time: 如果大于这个时间工作量未减少则退出工作
    """
  def run(self, same_work=False):
    """ 主函数, 负责一次完整的战斗.
        Args:
          same_work: 如果为 True, 则说明上一次战斗和当前执行的战斗相同并且上一次战斗正常退出,
用于简化操作流程
    """
```

注意 `exercise` 一般不调用 `run_for_times` 系列方法, 因为五次演习逻辑上被视为单次多点战斗.

start_scripts, DailyOperation

AutoWSGR.scripts.main.start_scripts
AutoWSGR.scripts.daily_api.DailyOperation

```
class DailyOperation():
    def __init__(self, setting_path) -> None:
```

```
        """
            setting_path: 用户配置文件路径
        """
    def run(self):
        """开始运行"""

def start_script(settings_path=None):
    """启动脚本,返回一个 Timer 记录器

    Returns:
        Timer: 该模拟器的记录器
    """
```

AutoWSGR.fight.decisive_battle.DecisiveBattle

```
class DecisiveBattle():
    """决战控制模块
    目前仅支持 E5, E6, E4
    """
    def run_for_times(self, times=1):
      """多次运行
      """


    def run(self):
      """单次运行
      """
      self.run_for_times()

    def __init__(self, timer: Timer, chapter=6, map=1, node='A', version=3,
                 level1=["鲃鱼", "U-1206", "U-47", "射水鱼", "U-96", "U-1405"],
                 level2=["U-81", "大青花鱼"],
                 flagship_priority=["U-1405", "U-47", "U-96", "U-1206"],
                 logic=None, *args, **kwargs):
        """初始化控制模块
        Important Information:
            : 决战逻辑相当优秀, 不建议自行改写, 满配情况约需要 3~6 快速修复打完一轮 E6, 耗时约为
25mins
            : 请保证可以直接使用上次选船通关, 暂时不支持自动选船
        Args:
            timer (Timer): 记录器
            chapter (int, optional): 决战章节,请保证为 [1, 6] 中的整数. Defaults to 6.
            map (int, optional): 当前小关. Defaults to 1.
            node (str, optional): 当前节点. Defaults to 'A'.
            version (int, optional): 决战版本. Defaults to 3.
            level1: 第一优先舰队, Defaults Recommend
            level2: 第二优先舰船, Defaults Recommend
            flagship_priority: 旗舰优先级, Defaults Recommend
            logic: 逻辑模块, unsupported since now
        Examples:
            : 若当前决战进度为还没打 6-2-A, 则应填写 chapter=6, map=1, node='A'
            : 可以支持断点开始
```

```
        """

    def buy_ticket(self, use='steel', times=3):
        """购买磁盘
          Args:
            use: 使用哪种资源 {"stell", "oil", "ammo", "aluminium"},
            times: 购买次数
        """
```

cook, build

AutoWSGR.game.game_operation.build
AutoWSGR.game.game_operation.cook

```
def build(timer:Timer, type, resources=None, fast=False, force=False):
    """建造操作
    Args:
        timer (Timer): _description_
        type (str): "ship"/"equipment"
        resources: 一个列表，表示油弹钢铝四项资源. Defaults to None.
        fast (bool, optional): 是否快速建造. Defaults to False.
        force (bool, optional): 如果队列已满，是否立刻结束一个以开始建造. Defaults to
False.
    """
def cook(timer:Timer, position:int):
    """食堂做菜
    Args:
        position (int): 第几个菜谱(1-3)
    """
```

Expedition

AutoWSGR.game.game_operation.Expedition

```
class Expedition:

    def __init__(self, timer: Timer) -> None:
        """
            timer: 控制器
        """
    def run(self, force=False):
        """检查远征，如果有未收获远征，则全部收获并用原队伍继续
        Args:
            force (bool): 是否强制检查
        Returns:
            bool: 是否进行了收远征操作
        """
```

启用基于 OCR 的高级功能

启用基于 OCR 的高级功能后, 能够支持更复杂的挂机策略以及自动化舰队决战等.

该功能要求分辨率在 `1280x720` 及以上，推荐使用 `1280x720`

配置舰船名称文件

启用基于舰船识别功能需要配置舰船名称文件, 我们提供了一个文件模板, 位于 `/src/AutoWSGR/data/default_ship_names.yaml`.

该模板中所有舰船名和官方图鉴保持一致, 当前更新到絮弗伦版本, 欢迎 PR.

如果某一舰船有多艘, 请使用以下格式填写:

```yaml
No.1: # 胡德
    - 胡德荣耀
    - 胡德飙车
    - 胡德未改
```

我们使用舰船名称唯一区分不同的舰船, 如果两艘舰船名字相同, 会认为她们是同一艘船, 请确保使用到的舰船不会出现重名.

在舰队决战功能中, 为了正确识别可选舰船, **"上次选船" 包含的所有舰船的名字都应该被添加到舰船名称文件.**

另外，由于 `easyocr` 工具本身可能没有收录一些中文字体, 所以无法识别部分生僻字, 经典例子是日系动物园和 "鲃鱼", 这里推荐把 "鲃鱼" 改名为 "肥鱼", 以后会解决这个问题.

常见报错

系统找不到指定文件

```
FileNotFoundError: [WinError 2] 系统找不到指定的文件。
```

这个报错是由于 Windows 或者杀毒软件将一些文件识别成了病毒并删除导致的, 请检查 Windows 病毒防护日志, 找到被删除的文件并加入白名单, 重新从项目库拉去对应文件解决问题.