概述

C++版ai位于example_cpp文件夹中。

python版ai位于example_py文件夹中。

以下是对ai运行流程、文件夹中各文件结构等的解释。

ai运行流程

1、ai启动后，会获得自己的阵营。此后会根据己方阵营选取卡组。

> 阵营为0的玩家先手，阵营为1的玩家后手。

> 在chooseCards()/choose_cards()函数中，玩家可以根据阵营做出自己的卡牌选择。

2、每位玩家的回合开始时，judger会向ai发送当前游戏局面信息(json格式)。
updateGameInfo()/update_game_info()函数会处理相关的json串并将游戏局面信息存储在类属性中。

> 存储游戏局面信息的结构体/类的具体结构可见gameunit.hpp/gameunit.py,对其的解读可见gameunit。

> 利用这些结构体/类存储的类属性可见AI类属性。

玩家根据当前游戏局面信息进行自己的处理，并调用相关的函数发送自己的操作。

> 目前相关的游戏操作函数包括summon、move、attack、use函数。

每次操作后，judger会向ai发送操作后的当前游戏局面信息。

> 在summon、move、attack、use函数中已包含更新信息的操作，玩家无需再次更新。

玩家可以不断发送自己的操作，直到发送结束回合的指令为止。

> 用于结束当前回合的函数是endRound()/self.end_round()函数。

> 一些其它的状况也可能导致回合结束，比如超时等。

2、非当前回合的ai将不会接收到任何信息，发出的信息也不会得到接收。

3、建议直接调用相关的函数进行消息的收发。您需要的做的仅仅是处理相关的数据。

> 不合法的信息发送可能导致回合结束/直接判负等后果。

注意AI的通信使用标准输入/输出流。除了直接使用sdk中相关的函数外，请不要在标准输出流中输出任何信息。

测试

使用judger启动游戏逻辑和ai

附带了可供玩家本地测试用的评测机和游戏逻辑文件（版本与实际版本略有差别）。

启动指令: python <judger路径> <启动逻辑command> <启动AI 1 command> <启动AI 2 command> <逻辑生成replay路径>

启动指令示例(在根目录下运行，注意确认路径是否正确)

```
python ./game/Judger/judger.py python+./game/main.py
python+./example/example_py/main.py  ./example/example_cpp/main  record
```

AI可以是C++版本，也可以选择python版本。选择其一即可。C++版本的AI需要使用对应的可执行文件，python版本的AI需要使用对应的.py文件。

（对于样例C++版本AI，可以使用`make`命令编译出对应的可执行文件。）

运行时，注意在工作路径下需要Data.json文件。实际评测时，（如果用到）只需确认压缩包里有Data.json文件即可。

运行结束后，会输出`{"end_info": "{\"0\": ..., \"1\": ...}", "record": [...]}`，其中end_info表示0、1号玩家的得分，record记录了每个回合AI的物理用时和状态。（得分=1000*对对手神迹造成的伤害+击杀对手生物总星数。掉线/超时AI得分会被清零，同时其对手得分加1。得分相同时会额外再给后手玩家加一分，判定后手玩家胜利。此得分仅表示游戏得分，与天梯积分无关。）

注意judger与AI的通信使用标准输入/输出流。除了直接使用sdk中相关的函数外，请不要在标准输出流中输出任何信息。

main.py中有一个DEBUG参数，设置为True的时候会在当前目录下生成一个log.txt文件，记录收发的信息。

example_cpp

此为对C++版本的AI的文件夹内的各文件的解释。

makefile

C++的makefile文件，目前仅适用于样例AI。玩家编写自己的AI后，需要根据实际情况修改makefile。

json.hpp

开源的json库。

需要调用该json库的地方已经给出相关代码，您不必关注相关细节。

gameunit.h & gameunit.cpp

包含若干结构体，用于表述游戏中各单位的信息。

> 具体的结构体的结构可查看gameunit板块。

card.h

包含生物、神器的基本属性。启动ai时会从Data.json中读取。

calculator.h & calculator.cpp

计算几何库。包含以下函数。您可以调用以下函数来进行相关的位置计算。

其中Point与gameunit::Pos类型相同。

```
std::vector<Point> search_path(Point start, Point to,
    std::vector<Point> obstacles={}, std::vector<Point> obstructs={})
```

通过起点、终点、阻挡列表、拦截列表寻找路径，obstacles代表不能停留也不能经过的点，obstructs代表可以停留但不能经过的点。

```
std::vector<Point> path(gameunit::Unit unit, Point dest, gameunit::Map
_map)
```

用A*算法给出从生物unit到dest点的路径（包含起点），无法给出路径时返回空数组。

```
int cube_distance(Point a, Point b)
```

给出两个位置之间的距离。

```
std::vector<std::vector<Point>> reachable(gameunit::Unit unit,
gameunit::Map _map)
```

给出生物unit在其最大步数内可达的位置，结果为数组，数组中的元素为经过等于下标的步数可达的位置的数组（例如result[1]为经过1步能到达的位置的数组）。

```
std::vector<gameunit::Unit> units_in_range(Point pos, int dist,
gameunit::Map _map, int camp=-1,
                                bool flyingIncluded=true, bool
onlandIncluded=true)
```

给出某点pos在给定范围内存在的生物,dist为给定的范围（步数），camp默认为-1，将会返回所有阵营的生物，0为先手阵营，1为后手阵营；flyingIncluded表示将飞行生物包含其中，onlandIncluded为将地面生物包含其中，默认两者都包含。

```
bool in_map(Point pos)
```

判断某个点pos是否在地图范围之内，返回布尔值。

```
std::vector<Point> all_pos_in_map()
```

给出地图内的所有点。

ai_client.h & ai_client.cpp

包含ai基类。封装好了部分通信函数和查找函数。

类属性

map

Map。

players

Player数组。长度为2。Player在数组中的下标与其camp相同。

round

int。当前游戏回合。

my_camp

int。己方阵营。

my_artifacts

string数组。用于确认己方神器。

my_creatures

string数组。用于确认己方生物。

> 注：神器与生物已经确定后便无法在游戏中修改。但可以通过访问my_artifacts与my_creatures快速获取己方所选神器与生物。

前一部分为与游戏通信的函数。建议您不要修改它们。

```
void updateGameInfo()
```

读入游戏信息并更新相关数据。

```
void chooseCards()
```

设置卡组，并调用init()函数确定初始卡组。

```
void play()
```

结合当前局面信息做出操作。

```
void init()
```

选择初始神器(my_artifacts)和生物(my_creatures)

```
void summon(std::string type, int level, int x, int y, int z)
```

在地图(**x**, **y**, **z**)处召唤一个本方类型为**type**，星级为**level**的生物

```
void summon(std::string, int level, std::vector<int> position)
```

```
void summon(std::string, int level, std::tuple<int, int, int> position)
```

在地图**position**处召唤一个本方类型为**type**，星级为**level**的生物

```
void move(int mover, int x, int y, int z)
```

将地图上id为**mover**的生物移动到地图(**x**, **y**, **z**)处。

```
void move(int mover, std::vector<int> position)
```

```
void move(int mover, std::tuple<int, int, int> position)
```

将地图上id为**mover**的生物移动到地图**position**处。

```
void attack(int attacker, int target)
```

令地图上id为**attacker**的生物攻击地图上id为**target**的生物或神迹。

```
void use(int artifact, int target)
```

对id为**target**的生物使用id为**artifact**的神器

```
void use(int artifact, std::vector<int> target)
```

```
void use(int artifact, std::tuple<int, int, int> target)
```

对**target**位置使用id为**artifact**的神器

```
void endRound()
```

结束当前回合。

除了以上与游戏通信相关的函数外，ai_client还包含以下可能有助于快速写AI的函数。这些函数仅在本地进行相关的查询操作。

```
int getDistanceOnGround(gameunit::Pos pos_a, gameunit::Pos pos_b, int
camp)
```

获取**camp**阵营生物从位置**pos_a**到位置**pos_b**的地面距离(不经过地面障碍或敌方地面生物)。

```
int getDistanceInSky(gameunit::Pos pos_a, gameunit::Pos pos_b, int camp)
```

获取**camp**阵营生物从位置**pos_a**到位置**pos_b**的飞行距离(不经过飞行障碍或敌方飞行生物)。

```
int checkBarrack(gameunit::Pos pos)
```

判定位置**pos**的驻扎情况。不是驻扎点返回-2,中立返回-1,否则返回占领该驻扎点的阵营(0或1)。

```
bool canAttack(gameunit::Unit attacker, gameunit::Unit target)
```

判断生物**attacker**能否攻击到生物**target**(只考虑攻击力、攻击范围)。

```
bool canUseArtifact(gameunit::Artifact artifact, gameunit::Pos pos, int
camp)
```

判断能否对位置**pos**使用神器**artifact**(不考虑消耗、冷却)。

```
bool canUseArtifact(gameunit::Artifact artifact, gameunit::Unit unit)
```

判断能否对生物**pos**使用神器**artifact**(不考虑消耗、冷却)。

```
gameunit::Unit getUnitByPos(gameunit::Pos pos, bool flying)
```

获取位置**pos**上的生物。

```
gameunit::Unit getUnitById(int unit_id)
```

获取上id为**unit_id**的生物。

```
std::vector<gameunit::Unit> getUnitsByCamp(int unit_camp)
```

获取所有阵营为**unit_camp**的生物。

```
std::vector<gameunit::Pos> getSummonPosByCamp(int camp)
```

获取所有属于阵营**camp**的出兵点(初始出兵点+额外出兵点)。

ai.cpp

包含类AI。该类继承自AiClient类。玩家需要在AiClient类的基础上，重写play()函数及其它函数，完成自己的AI。

同时该文件中也包含main()函数，用于启动AI。

ai-sample.cpp

一个样例AI，玩家可以用于参考。

example_py

此为对python版本的AI的文件夹内的各文件的解释。

gameunit.py

包含若干类，用于表述游戏中各单位的信息。

> 具体的类的结构可查看gameunit板块。

card.py

包含生物、神器的基本属性。会从Data.json中读取。

calculator.py

计算几何库。包含以下函数。您可以调用以下函数来进行相关的位置计算。

```
search_path(start, end, obstacles, obstructs)
```

通过起点、终点、阻挡列表、拦截列表寻找路径，obstacles代表不能停留也不能经过的点，obstructs代表可以停留但不能经过的点。

```
path(unit, dest, _map)
```

用A*算法给出生物unit到位置dest的路径（包含起点）。无法给出路径时返回False。

```
cube_distance(pos1, pos2)
```

给出两个位置之间的距离。

```
reachable(unit, _map)
```

给出生物unit在其最大步数内可达的位置，结果为列表，列表中的元素为经过等于下标的步数可达的位置的列表（例如result[1]为经过1步能到达的位置的列表）。

```
units_in_range(pos, dist, _map, camp=-1, flyingIncluded=True,
onlandIncluded=True)
```

给出某点pos在给定范围内存在的生物,dist为给定的范围（步数），camp默认为-1，将会返回所有阵营的生物，0为先手阵营，1为后手阵营；flyingIncluded表示将飞行生物包含其中，onlandIncluded为将地面生物包含其中，默认两者都包含。

```
in_map(pos)
```

判断某个点pos是否在地图范围之内，返回布尔值。

```
all_pos_in_map()
```

给出地图内的所有点。

ai_client.py

包含ai基类。封装好了部分通信函数和查找函数。

类属性

map

Map。

players

Player数组。长度为2。Player在数组中的下标与其camp相同。

round

int。当前游戏回合。

my_camp

int。己方阵营。

my_artifacts

string数组。用于确认己方神器。

my_creatures

string数组。用于确认己方生物。

> 注：神器与生物已经确定后便无法在游戏中修改。但可以通过访问my_artifacts与my_creatures快速获取己方所选神器与生物。

前一部分为与游戏通信的函数。建议您不要修改它们。

```
update_game_info()
```

读入游戏信息并更新相关数据。

```
choose_cards()
```

设置卡组，并调用init()函数确定初始卡组。

```
play()
```

结合当前局面信息做出操作。

```
 init()
```

选择初始神器(my_artifacts)和生物(my_creatures)。

```
 summon(_type, level, position)
```

在**position**处召唤一个本方类型为**_type**，星级为**level**的生物。

```
move(mover, position)
```

将id为**mover**的生物移动到**position**处。

```
attack(attacker, target)
```

令id为**attacker**的生物攻击id为**target**的生物或神迹。

```
use(artifact, target)
```

对id为**target**的生物/**target**位置使用id为**artifact**的神器。

```
end_round()
```

结束当前回合。

除了以上与游戏通信相关的函数外，ai_client还包含以下可能有助于快速写AI的函数。这些函数仅在本地进行相关的查询操作。

```
get_distance_on_ground(pos_a, pos_b, camp)
```

获取**camp**阵营生物从位置**pos_a**到位置**pos_b**的地面距离(不经过地面障碍或敌方地面生物)。

```
get_distance_in_sky(pos_a, pos_b, camp)
```

获取**camp**阵营生物从位置**pos_a**到位置**pos_b**的飞行距离(不经过飞行障碍或敌方飞行生物)。

```
check_barrack(pos)
```

判定位置**pos**的驻扎情况。不是驻扎点返回-2,中立返回-1,否则返回占领该驻扎点的阵营(0或1)。

```
can_attack(attacker, target)
```

判断生物**attacker**能否攻击到生物**target**(只考虑攻击力、攻击范围)。

```
can_use_artifact(artifact, target, camp)
```

判断阵营**camp**的玩家能否对目标**target**使用神器**artifact**(不考虑消耗、冷却)。

```
get_unit_by_pos(_map, pos, flying)
```

获取位置**pos**上的生物。无生物时返回None。

```
get_unit_by_id(unit_id)
```

获取id为**unit_id**的生物。无生物时返回None。

```
get_units_by_camp(_map, unit_camp)
```

获取所有阵营为**unit_camp**的生物。无生物时返回空数组。

```
get_summon_pos_by_camp(camp)
```

获取所有属于阵营**camp**的出兵点(初始出兵点+额外出兵点)。

ai.py

包含类AI。该类继承自AiClient类。玩家需要在AiClient类的基础上，重写play()函数及其它函数，完成自己的AI。

同时该文件中也包含main()函数，用于启动AI。

main.py

（即ai-sample）一个样例AI，玩家可以用于参考。

gameunit

ai回合开始时/每次执行非结束回合的游戏操作时会收到一个表述游戏当前局面信息的json格式的字符串。对该字符串的解析已于updateGameInfo()函数中实现。(gameunit.h中的from_json()用于把json转化成相应对象，细节此处不表。)updateGameInfo()函数会将相关的信息以结构体的形式存入类属性中。相关的结构体在gameunit.h中给出。以下为gameunit.h相关结构体的具体参数的解释。

python版本的gameunit与C++版本的gameunit的设计完全相同（不过由于类型上的不同，C++的vector对应python的list，C++的std::tuple<>对应python的tuple，等），此处不再赘述。

以下叙述中Pos类型表示std::tuple<int, int, int>，用于表示位置。

Unit

表示单个生物。

id

int。表示生物的id。

camp

int。表示生物所属阵营。

type

string。表示生物的种类。有"Archer", "Swordsman", "BlackBat", "Priest", "VolcanoDragon", "Inferno", "FrostDragon"七种取值。

cost

int。表示生物的法力消耗。

atk

int。表示生物的攻击。

max_hp

int。表示生物的生命上限。

hp

int。表示生物的当前生命。

atk_range

二元int数组。表示生物攻击最小范围/最大范围。

> （攻击范围0表示不可攻击/反击，0-0表示可攻击与自身相同格子的生物）

max_move

int。表示生物的行动力。

cool_down

int。表示生物的冷却。

pos

Pos。表示生物的位置。

level

int。表示生物的等级。

flying

bool。表示生物是否飞行。

atk_flying

bool。表示生物能否攻击飞行生物。

agility

bool。表示生物是否迅捷。

holy_shield

bool。表示生物是否具有圣盾。

can_atk

bool。表示生物能否攻击。

can_move

bool。表示生物能否移动。

Barrack

表示一个驻扎点。

pos

Pos。表示驻扎点的位置。

camp

int。表示驻扎点所属阵营。

summon_pos_list

Pos数组。表示驻扎点所控制的出兵点的位置。

Miracle

表示一个神迹。

camp

int。表示神迹所属阵营。

max_hp

int。表示神迹生命上限。

hp

int。表示神迹生命值。

pos

Pos。表示神迹的位置。

summon_pos_list

Pos数组。表示神迹控制的初始出兵点的位置。

id

int。表示神迹的id。

Obstacle

表示一个地图障碍。

type

string。表示障碍的种类。

pos

Pos。表示障碍的位置。

allow_flying

bool。表示障碍是否允许飞行生物通过。

allow_ground

bool。表示障碍是否允许地面生物通过。

Artifact

表示一个神器。

id

int。表示神器的id。

name

string。表示神器的名字。有"HolyLight", "SalamanderShield", "InfernoFlame", "WindBlessing"四种取值。

camp

int。表示神器所属的阵营。

cost

int。表示神器的法力消耗。

max_cool_down

int。表示神器的最大冷却时间。

cool_down_time

int。表示神器的目前冷却时间。

state

string。表示神器的目前使用状态。有"Ready", "In Use", "Cooling Down"三种取值。

target_type

string。表示神器使用对象的种类。有"Unit", "Pos"两种取值。

last_used_pos

Pos。表示神器上次使用对象的位置。默认值为(-1, -1, -1)。（target_type为"Unit"时，表示使用时对象的位置。）

CreatureCapacity

生物召唤情况。

type

string。表示生物种类。有"Archer", "Swordsman", "BlackBat", "Priest", "VolcanoDragon", "Inferno", "FrostDragon"七种取值。

available_count

int。表示生物的生物槽容量。

cool_down_list

int数组。表示生物的冷却时间。

Map

units

Unit数组。包括地图上所有生物。

barracks

Barrack数组。包括地图上所有驻扎点。

miracles

Miracle数组。包括地图上所有神迹。

obstacles

Obstacle数组。包括地图上所有障碍。

flying_obstacles

Obstacle数组。包括地图上所有飞行障碍。

ground_obstacles

Obstacle数组。包括地图上所有地面障碍。

Player

表示一个玩家。

camp

int。表示玩家所处的阵营。

artifact

Artifact数组。表示玩家所拥有的神器的情况。

mana

int。表示玩家当前法力值。

max_mana

int。表示玩家最大法力值。

creature_capacity

CreatureCapacity数组。表示生物的召唤情况。

newly_summoned_id_list

int数组。表示玩家最新召唤的生物的id。