2022 种子杯赛题

> 这部分属于客户端(client)的python代码实现，用于和服务端(server)交互。client向server发送玩家的动作信息，并接收server返回的地图信息。

##目录结构

```
client/python
├── base.py     # package base
├── config.py   # config loader
├── logger.py   # logger config
├── main.py     # terminal playing API
├── README.md
├── req.py      # request package
├── resp.py     # response package
└── ui.py       # graphic user interface
```

使用说明

首先运行server，接着启动client。main.py中提供了termPlayAPI接口，用于在命令行进行游戏，选手可以参照这部分的代码编写程序

首先体验一下手动和机器人(bot)进行对战的过程

```
# launch server
cd bin
./server # if run into permission denied problem, run `chmod +x server`
first

# launch bot
cd bin
./silly-bot # if run into permission denied problem, run `chmod +x server`
first

# launch python client
cd client/python
python main.py
```

开发指导

前面已经提过，client负责向server发送玩家的动作，并接收server返回的地图信息。

总体流程：

游戏运行的过程中client负责向server发送玩家的动作信息，即玩家当前回合采取了哪个动作。server收到动作信息后会进行刷新游戏状态并在**回合结束时**返回地图的状态信息，即当前地图上玩家的位置，炸弹的位置等，

若当前回合为最终回合（游戏达到了最大回合数或有一个玩家胜出），则server会返回游戏结束的信息。

具体的流程如下所示：

1. 使用client连接server；
2. client向server发送初始化请求包；
3. client接收server回复的信息；
4. 若
    1. 3中的server返回了游戏结束的信息，则游戏结束；
    2. 3中的server返回地图状态信息，表示游戏未结束，则选手可以解析地图的状态并做出当前玩家的动作决策，并使用client向server发送要执行的动作。
5. 重复步骤3、4。

使用client连接server：

首先确保server端正常运行，接着创建client对象：

```
with Client() as client:
    client.connect()
```

如果连接成功，则将弹出提示信息；若连接失败，则程序会自动退出。

使用client向server发送初始化信息：

发送初始化信息的时候需要在InitReq中输入你的玩家名称，再使用PacketReq创建初始化请求包，最后使用client发送初始化请求。初始化请求主要用于告诉server端有新的玩家加入。

```
initPacket = PacketReq(PacketType.InitReq, InitReq("<your name>"))
client.send(initPacket)
```

使用client接收server返回的信息：

下面是使用client接收server返回信息并处理的代码逻辑示例，

```
resp = client.recv() # recieve response from server

if resp.type == PacketType.ActionResp:
    action = parseRespAndTakeAction(resp)

elif resp.type == PacketType.GameOver:
    print("game over!")

else:
    logger->error("unknow response type")
    exit(-1)
```

其中resp为class PacketResp类型，class PacketResp的定义在resp.py中，具体定义如下：

```
class PacketResp(JsonBase):
    def __init__(
        self,
        type: PacketType = PacketType.ActionResp,
        data: Union[ActionResp, GameOverResp],
    ) -> None:
        super().__init__()
        self.type = type
        self.data = data

    def from_json(self, j: str):
        ...
```

其中type字段用来表示回复的类型，data字段用来表示具体的信息，from_json函数是用来解析数据包的，选手们不需要关注，想了解的也可以仔细阅读代码。

当type为PacketType.ActionResp时，data字段为class ActionResp，包含地图的状态信息，选手们可以根据这部分信息采取动作决策。class ActionResp的具体定义如下（在resp.py中）：

```
class ActionResp(JsonBase):
    def __init__(
        self,
        player_id: int = 0,
        round: int = 0,
        map: List[Map] = [],
    ) -> None:
        super().__init__()
        self.player_id = player_id
        self.round = round
        self.map = map
```

其中player_id是己方玩家的id，round是游戏当前的回合数，map是一个列表，包含了地图的信息，其中每一个元素代表地图上一个格子的信息。若地图的大小为 15×15 ，则map的大小为 $225 = 15 \times 15$ ，其中地图第 i 行第 j 列（从 0 开始数）的格子信息位于map中下标为 $i \times 15 + j$ 的位置。map列表中的每一个元素为class Map类型，其具体定义如下（在resp.py中）：

```
class Map(JsonBase):
    def __init__(
        self,
        x: int = 0,
        y: int = 0,
        last_bomb_round: int = -1,
        objs: List[Obj] = [],
    ) -> None:
        super().__init__()
        self.x = x
```

```
        self.y = y
        self.last_bomb_round = last_bomb_round
        self.objs = objs
```

其中x y这两个字段标识该格子的位置，和上面所说的 i 与 j 同义。 last_bomb_round为该格子的上一次被炸的回合数，主要判断GUI是否显示爆炸特效💥。objs是一个列表，包含了这个格子上含有的所有objects（玩家、炸弹以及道具等）。objs列表中的每一个元素 class Obj类型，其具体定义如下（在resp.py中）：

```
class Obj(JsonBase):
    def __init__(
        self,
        type: ObjType = ObjType.Null,
        property: Union[None, Player, Bomb, Block, Item] = None,
    ) -> None:
        super().__init__()
        self.type = type
        self.property = property
```

type字段标识物体的类型，物体的类型如下所示：

```
class ObjType(JsonIntEnum):
    Null = 0    # no object
    Player = 1  # 玩家（不区分敌我）
    Bomb = 2    # 炸弹
    Block = 3   # 障碍物
    Item = 4    # 道具
```

property为物体的具体信息，其类型可能是None, Player, Bomb, Block, Item中的任意一种，类型包含的具体字段不再赘述，选手可以在resp.py中自行查看相关信息。

使用client向server发送动作信息：

玩家总共可以采取6种动作，动作的定义位于req.py中

```
class ActionType(JsonIntEnum):
    """Action space."""
    SILENT = 0      # keep slient
    MOVE_LEFT = 1
    MOVE_RIGHT = 2
    MOVE_UP = 3
    MOVE_DOWN = 4
    PLACED = 5      # place bomb
```

如果希望玩家玩家采取向左移动的动作，则使用以下代码：

```
action = ActionReq(<your_player_id>, ActionType.MOVE_LEFT)
actionPacket = PacketReq(PacketType.ActionReq, [action])
client.send(actionPacket)
```

在一个回合内，可以做出多个行动，行动的最大个数为玩家的行动速度（初赛中为行动速度为2且不变）。若要做出多个行动，不需要发送多个PacketReq，只需要将创建PacketReq的代码改为如下代码即可：

```
actionPacket = PacketReq(PacketType.ActionReq, [action0, action1])
```

其中，第二个变量变为了一个列表，列表内是多个ActionReq。