数据协议

```
message Observation {
  FrameState frame_state = 1;      // 局部环境数据
  ScoreInfo score_info = 2;        // 得分信息
  repeated MapInfo map_info = 3;   // 局部地图信息
  repeated int32 legal_act = 4;    // 合法动作
}

// 地图信息为二维信息
message MapInfo {
  repeated int32 values = 1; // 地图信息行信息
}

message ScoreInfo {
  float score = 1;                     // 即时得分
  float total_score = 2;               // 总得分
  int32 step_no = 3;                   // 步号
  int32 treasure_collected_count = 4; // 收集到的宝箱数量
  int32 treasure_score = 5;            // 收集到的宝箱得分
  int32 buff_count = 6;               // 收集到的buff数量
  int32 talent_count = 7;             // 使用技能的数量
}

message ExtraInfo {
  int32 result_code = 1;       // 错误码
  string result_message = 2;   // 错误信息
  FrameState frame_state = 3; // 全局环境数据
  GameInfo game_info = 4;         // 环境信息
}

// 相对位置信息
message RelativePosition {
  RelativeDirection direction = 1;  // 相对方位（离散化）
  RelativeDistance l2_distance = 2; // L2距离（离散化）
}

// 离散化的相对方向
enum RelativeDirection {
  RELATIVE_DIRECTION_NONE = 0;
  East = 1;
  NorthEast = 2;
  North = 3;
  NorthWest = 4;
  West = 5;
  SouthWest = 6;
  South = 7;
  SouthEast = 8;
}
```

```
// 离散化的相对距离
// 每个级别相差20个网格坐标距离（L2距离）
enum RelativeDistance {
  RELATIVE_DISTANCE_NONE = 0;
  VerySmall = 1;  //0 ~ 20
  Small = 2;      //20 ~ 40
  Medium = 3;     //40 ~ 60
  Large = 4;      //60 ~ 80
  VeryLarge = 5;  //80 ~ 180
}


// 环境数据
message FrameState {
  int32 step_no = 1;              // 步数
  repeated RealmHero heroes = 2;  // 英雄状态
  repeated RealmOrgan organs = 3; // 物件状态
}

message GameInfo {
  float score = 1;                    // 即时得分
  float total_score = 2;              // 总得分
  int32 step_no = 3;                  // 步号
  Position pos = 4;                   // 英雄当前位置
  Position start_pos = 5;             // 起点位置
  Position end_pos = 6;               // 终点位置
  int32 treasure_collected_count = 7; // 收集到的宝箱数量
  int32 treasure_score = 8;           // 收集到的宝箱得分
  int32 treasure_count = 9;           // 宝箱总数
  int32 buff_count = 10;              // 收集到的buff数量
  int32 talent_count = 11;            // 使用技能的数量
  int32 buff_remain_time = 12;        // 剩余加速时间
  int32 buff_duration = 13;           // 总加速时间
  repeated MapInfo map_info = 14;     // 局部地图信息
  repeated int32 obstacle_id = 15;    // 障碍物id
}

// 英雄信息
message RealmHero {
  int32 hero_id = 1;  // 英雄id
  Position pos = 2;   // 英雄当前位置
  int32 speed_up = 3; // 英雄是否处于加速状态
  Talent talent = 4;  // 召唤师技能
  int32 buff_remain_time = 5;         // 剩余加速时间
}

// 召唤师技能
message Talent {
  int32 talent_type = 1; // 技能名
  int32 status = 2;      // 技能状态, 0表示CD中, 1表示available
  int32 cooldown = 3;    // 技能剩余冷却时间
}
```

```
message RealmOrgan {
  int32 sub_type = 1; // 物件类型, 1代表宝箱, 2代表加速buff,3代表起点,4代表终点
  int32 config_id = 2; // 物件id 0代表buff, 1~13代表宝箱 21代表起点, 22代表终点
  int32 status = 3; // 0表示不可获取, 1表示可获取, -1表示视野外
  Position pos = 4; // 物件位置坐标
  int32 cooldown = 5;                 // 物件剩余冷却时间
  RelativePosition relative_pos = 6; // 物件相对位置
}

message Position {
  int32 x = 1; // x坐标
  int32 z = 2; // z坐标
}
```

1. 闪现的加入

在基础的代码中，agent无法使用闪现，故需要增加该功能。

首先在文件 `conf/conf.py` 中将特征 `FEATURES` 的动作部分增加8个方向，同时增设 `DIM_OF_ACTION = 16` 表示共有16维的动作输出(注意在 `feature/definition.py` 中需要的 `NummpyData2SampleData` 需要同样进行修改)：

```
# conf/conf.py
FEATURES = [
    2,
    6,
    6,
    8 + 8, # 动作方向
    4, # 奖励信息
]
DIM_OF_ACTION = 16
SAMPLE_DIM = 2 * (DIM_OF_OBSERVATION + DIM_OF_ACTION) + 4
```

在 `algorithm/algorithm.py` 中也需要改为 `self.act_shape = Config.DIM_OF_ACTION`

随后进入特征处理部分，加入闪现的cd，处理合法动作访问，在 `Preprocessor.py` 中加入如下变量

```
def __init__(self):
    self.max_map_dist = math.hypot(128, 128) # 最大距离，用于归一化闪现距离
    self.flash_range = 16.0 # 根据文档闪现距离为16

def __reset(self):
    self.is_flashed = True  # 每次reset时初始化为冷却已好
```

在 `pb2struct` 函数中处理冷却标记以及cd具体时间：

```
# 见数据协议
self.flash_cd = hero["talent"]["status"]
if self.flash_cd == 1:
```

```
    self.is_flashed = True
else:
    self.is_flashed = False
```

在 `process` 中处理闪现的距离，前后位置等信息，用于奖励的设计以及模型的使用

```
end_dist = self.feature_end_pos[-1]  # 终点距离
r_flash = self.flash_range / self.max_map_dist  # 归一化距离
flash_used = (last_action >= Config.DIM_OF_ACTION_DIRECTION)  # 是否用闪现
d_after = max(0.0, end_dist - r_flash) if flash_used else end_dist  # 距离终点距离的减少
extra_feats = np.array([r_flash, end_dist, d_after, self.flash_cd / 100], dtype=np.float32) # 返回特征
```

在 `get_legal_action` 中需要将 `legal_action` 字段扩展成16维的bool数组，根据 `move_usable`， `is_flashed` 和 `flash_cd` 进行判断

最后设计 `reward_process`，如下：

```
# 传入d_before, d_after, flash_uesd
# d_before = end_dist
flash_cost = -0.02 if flash_used else 0.0
flash_gain = 0.0
flash_fail = 0.0
if flash_used and d_before is not None and d_after is not None:
    flash_gain = 0.1 * max(0.0, (d_before - d_after))
    flash_fail = -0.05 if d_after >= d_before else 0.0
```

终点奖励设计

为了使agent能更好的进入终点，并且防止在终点附近徘徊，考虑设计如下的奖励：

```
cone_reward = 0.3 * (0.3 - end_dist) if end_dist < 0.3 else 0.0
# 终点附近奖励，接近终点奖励提升，引导agent接近终点
end_success = 1.0 if end_dist < 1e-3 else 0.0  # 终点强奖励
```

Anti-Stuck

记录最近8帧的位置，如果没有进展则进行惩罚

首先定义在 `preprocessor.py` 的初始化函数中，并且在 `reset` 时进行清零

```
def __init__(self):
    self.pos_hist_window = 8  # 检测窗口：最近8帧
    self.no_progress_penalty = 0.2  # 没有前进，进行惩罚
    self.loop_penalty = 0.15  # 循环往复惩罚
    self.__pos_history = []  # 缓冲最新的位置
```

随后每次 `process()` 时将当前坐标加入 `_pos_history` 中，分两种情况处理：

- 若没有前进，则进行惩罚
- 若循环徘徊，最近4帧出现A-B-A-B的情况，进行惩罚

最终返回 `stuck_penalty` 给 `reward_process` 即可

```
self._pos_history.append(self.cur_pos)
if len(self._pos_history) > self.pos_hist_window:
    self._pos_history.pop(0)

    stuck_penalty = 0.0

# 原地不动
if len(self._pos_history) == self.pos_hist_window and
len(set(self._pos_history)) == 1:
    stuck_penalty -= self.no_progress_penalty

# 循环情况
if len(self._pos_history) >= 4:
    if (self._pos_history[-1] == self._pos_history[-3] and
        self._pos_history[-2] == self._pos_history[-4]):
        stuck_penalty -= self.loop_penalty
```

TTL机制

撞墙后的动作在若干步内被屏蔽，避免反复撞击

在 `preprocessor.py` 中加入 `bad_moves` 列表，以及相应的TTL屏蔽数值

```
def __init__(self):
    self.bad_moves      = {}       # {move_id: ttl}
    self.BAD_TTL_MOVE   = 10       # 普通移动动作的TTL
    self.BAD_TTL_FLASH  = 3        # 闪现动作的TTL

# 注意在reset中进行初始化清零
def reset(self):
    self.bad_moves.clear()
```

需要结合前面的闪现机制等，对函数 `get_legal_action` 进行更新，首先对被禁用动作的TTL减一(归零后可以重新使用)

```
for k in list(self.bad_moves.keys()):
    self.bad_moves[k] -= 1
    if self.bad_moves[k] <= 0:
        del self.bad_moves[k]
```

随后判断是否位置发生变化，如果没有认为发生撞墙，加入 `bad_moves` ，并且屏蔽动作

```
pos_unchanged = (
    abs(self.cur_pos_norm[0]-self.last_pos_norm[0])<1e-3 and
    abs(self.cur_pos_norm[1]-self.last_pos_norm[1])<1e-3
)
if pos_unchanged and 0 <= self.last_action < 16:
    if self.last_action < 8:
        self.bad_moves[self.last_action] = self.BAD_TTL_MOVE
    else:
        self.bad_moves[self.last_action % 8] = self.BAD_TTL_FLASH
```

```
for move_id in self.bad_moves.keys():
    legal_action[move_id] = False
```

两个保证策略：

- 终点已经找到时，强制解锁方向
- 如果全部动作都被屏蔽，则解决上一步的动作

```
# 终点策略
if self.is_end_pos_found:
    dx = self.end_pos[0] - self.cur_pos[0]
    dz = self.end_pos[1] - self.cur_pos[1]
    if max(abs(dx), abs(dz)) <= 1:
        theta = (math.degrees(math.atan2(dz, dx)) + 360) % 360
        dir_idx = int(((theta + 22.5) % 360) // 45)
        legal_action[dir_idx] = True

# 全零兜底退回
if not any(legal_action):
    fallback = self.prev_action_dir if self.prev_action_dir is not None
else 0
    legal_action[fallback] = True
```

领域与方向优化

1. 在 `_get_pos_feature` 函数中，加入位置感知，通过原来的相对向量转换成方向(0-7)，8维的数组

```
# 注意需要在config中修改，返回14维的位置特征
direction_onehot = np.zeros(8, dtype=np.float32)
if dist > 1e-4:
    theta = (math.degrees(math.atan2(relative_pos[1], relative_pos[0])) +
360) % 360
    dir_idx = int(((theta + 22.5) % 360) // 45)  # 0~7
    direction_onehot[dir_idx] = 1.0
```

2. 加入agent周围的3*3的领域可行特征，首先构造9个便宜(dx, dz)，根据工具函数 `_is_free` 判断是否有障碍，最终得到一个9维的bool数组

```
cur_cell = tuple(map(int, self.cur_pos))
offsets  = list(itertools.product([-1, 0, 1], repeat=2))
walkable = [
    1.0 if self._is_free((cur_cell[0]+dx, cur_cell[1]+dz)) else 0.0
    for dx, dz in offsets
]
```

3. 射限距离特征，取出上一步的动作的方向，沿着射限走20步，遇到边界/障碍则返回step/20，否则返回1

```
fwd_dir = (self.last_action % 8) if 0 <= self.last_action < 16 else None
ray_feat = [
```

```
    self._cast_ray(self.cur_pos, self._dir_lookup[fwd_dir])
    if fwd_dir is not None else 1.0
]
```

重复访问惩罚与直线奖励

1. `visit_counter` 定义在 `__init__` 中，同时注意在 `reset` 中清空，用于记录本格子被访问的次数，没进入一回合+1，限制最多惩罚5次

```
# 返回visit_penalty给reward_process
self.visit_counter[cur_cell] += 1
visit_penalty = -0.03 * min(self.visit_counter[cur_cell], 5)
```

2. 计算上一次移动的方向与本次的，映射转换成角度

```
turn_angle = 0.0
if self.prev_action_dir is not None and 0 <= self.last_action < 8:
    diff = abs(self.last_action - self.prev_action_dir) % 8
    diff = 8 - diff if diff > 4 else diff            # 归一到 0-4 步
    turn_angle = diff * 45                           # 转向角度:
0°,45°,…,180°
if 0 <= self.last_action < 8:
    self.prev_action_dir = self.last_action
```

在 `reward_process` 中使用 `turn_angle`，对走直线进行奖励，对转向惩罚

```
turn_penalty   = -0.002 * (turn_angle / 90.)
straight_bonus = 0.002 if turn_angle == 0 else 0.0
```

问题与改进

1. `_is_free` 函数需要进一步设计，考虑视野问题
2. 各种参数大小的调整
3. `Anti-stuck` 中考虑拓展情况