环境介绍

一、任务概述

峡谷漫步（GorgeWalk）是一个基于网格地图的导航任务，智能体需要学习在64×64的网格世界中从起点导航到终点，同时避开障碍物并可选择性地收集宝箱以获得额外积分。

1. 地图配置

- **网格尺寸**：64×64的二维网格空间，每个格子代表一个可移动单元
- **地图元素**：包含起点(START)、终点(END)、通路(PATH)、障碍物(OBSTACLE)和宝箱(TREASURE)五种元素
- **元素信息**：起点(START)、终点(END)、通路(PATH)、障碍物(OBSTACLE)位置固定，共计有10个宝箱，每次位置随机刷新。宝箱位置**完全随机生成**，在每次`reset()`时从所有可通行的PATH格子中随机选择位置。环境内置10个可能的宝箱点位，具体位置每局不同
- **移动规则**：智能体的动作方向有四个：上下左右；如果按照给定方向，智能体的下一个格子是通路，智能体正常前进一格，若是障碍物，则智能体的位置不变。
- **任务目标**：智能体在尽可能短的时间内，收集尽可能多的宝箱的同时抵达终点以获得最大总积分。

2. 地图元素编码

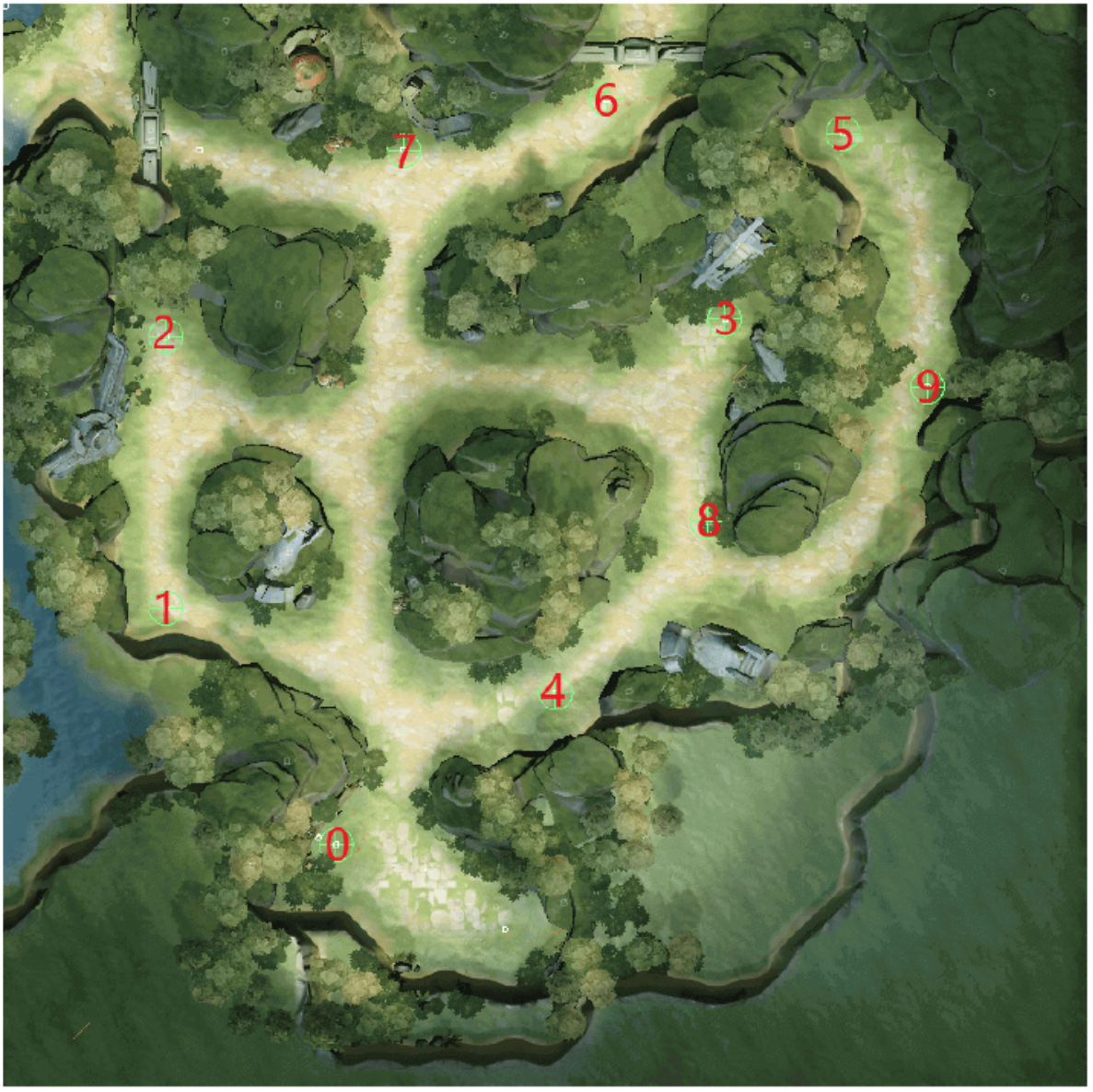
```
OBSTACLE = 0  # 障碍物
PATH = 1      # 可通行路径
START = 2     # 起点
END = 3       # 终点
TREASURE = 4  # 宝箱
```

3. 总积分计算

```
总积分 = 终点积分 + 步数积分 + 宝箱积分
```

- **终点积分**：到达终点获得150分
- **步数积分**：`(最大步数 - 完成步数) × 1.0
- **宝箱积分**：每个宝箱100分
- **注意**：若未在最大步数（1000）内到达终点，总积分归零

下图为某一次随机下，宝箱的分布与地图的状态。



4. 强化学习环境介绍

我们的环境除了适配各种决策智能体外，也是一个可以适配强化学习训练的环境。

强化学习是机器通过与环境交互来实现目标的一种计算方法。机器和环境的一轮交互是指，机器在环境的一个状态下做一个动作决策，把这个动作作用到环境当中，这个环境发生相应的改变并且将相应的奖励反馈和下一轮状态传回机器。这种交互是迭代进行的，**目标是最大化在多轮交互过程中获得的累积奖励（reward）的期望**。强化学习用智能体（agent）这个概念来表示做决策的机器。相比于有监督学习中的“模型”，强化学习中的“智能体”强调机器不但可以感知周围的环境信息，还可以通过做决策来直接改变这个环境，而不只是给出一些预测信号。reward的具体设置是人为设定的，以此引导智能体学到好的策略，该过程称为奖励塑形（reward shaping）。

在每一轮交互中，智能体感知到环境目前所处的状态（observation），经过自身的计算（choose_action）给出本轮的动作（action），将其作用到环境中（env.step）；环境得到智能体的动作后，产生相应的即时奖励信号（reward）并发生相应的状态转移。智能体则在下一轮交互中感知到新的环境状态，依次类推：

[**env.reset()**--获得初始observation]-->[**agent.choose_action()**--产生action]-->[**env.step(action)**--获得新的observation]-->[**agent.choose_action()**---产生action]-->[**env.step(action)**--获得新的observation] ···· -->[触发terminate或truncate信号，结束]

环境使用**openai gym接口**。在使用前要先重置（env.reset），环境执行一步为 env.step。函数具体的返回参数可以参考以下函数，其中observation表示目前所处的状态，reward表示目前获得的奖励，terminated表示目前是否能到达终点，truncated表示是否超时（即是否能在规定的步数内到达终点），info表示环境中的其他所有信息。其中包含内容可以参考文档后续内容。

env.reset()函数用于初始化环境信息，一般在训练最开始时使用。

```
observation, info = env.reset()
```

在本次测试的实验环境中env.step()函数代表对环境进行一轮更新，亦可理解为强化学习过程中的一轮交互。

```
observation, reward, terminated, truncated, info = env.step(action)
```

可在framework/trainer.py文件中具体查看智能体（agent）如何实现与环境的交互，以及训练的过程的具体内容。

若希望了解更多的强化学习知识，可以询问AI或者参考以下链接：

> https://hrl.boyuai.com/

二、观测空间

环境提供丰富的观测信息，采用`gym.spaces.Dict`结构：

观测项	类型	维度	说明
`local_view`	Box(int32)	(5, 5)	以智能体为中心的5×5局部视野
`agent_pos`	Box(int32)	(2,)	智能体当前坐标[x, y]
`target_pos`	Box(int32)	(2,)	终点坐标
`treasure_status`	MultiBinary	(10,)	10个宝箱的收集状态（1=未收集，0=已收集）
`location_memory`	Box(float32)	(64, 64)	位置访问记忆矩阵，每访问一次增加0.1
`map_visible`	Box(int32)	(64, 64)	已探索区域的地图信息（未探索区域为-1）

具体来说他的格式：

1. 单环境返回格式

`env.reset()`和`env.step()`返回的`observation`是一个字典，包含以下6个键：

```
observation = {
    "local_view": np.array(shape=(5, 5), dtype=np.int32),
    # 以智能体为中心的5×5局部视野
    # 值域: 0=障碍物, 1=通路, 2=起点, 3=终点, 4=宝箱
```

```
    "agent_pos": np.array(shape=(2,), dtype=np.int32),
    # 智能体当前位置 [x, y]

    "target_pos": np.array(shape=(2,), dtype=np.int32),
    # 终点位置 [x, y]

    "treasure_status": np.array(shape=(10,), dtype=np.int8),
    # 10个宝箱的状态，1=未收集，0=已收集

    "location_memory": np.array(shape=(64, 64), dtype=np.float32),
    # 位置访问记忆，每访问一次增加0.1，上限1.0，未访问过的位置均为0.0

    "map_visible": np.array(shape=(64, 64), dtype=np.int32),
    # 已探索区域的地图，探索过的区域显示地图元素，未探索为-1
    # 智能体当前位置标记为5
}
```

2. 异步并行环境返回格式

为了训练加速，推荐使用`gym.vector.AsyncVectorEnv`创建N个并行环境，环境并行的方法在示例代码中会给出（详见开发指南）；在创建并行环境时，所有返回值在最外层增加一个batch维度：

```
# 假设NUM_ENVS = 32
obs_batch = vec_env.reset()[^0]  # 返回元组的第一个元素

# 此时obs_batch的结构：
obs_batch = {
    "local_view": np.array(shape=(32, 5, 5), dtype=np.int32),
    "agent_pos": np.array(shape=(32, 2), dtype=np.int32),
    "target_pos": np.array(shape=(32, 2), dtype=np.int32),
    "treasure_status": np.array(shape=(32, 10), dtype=np.int8),
    "location_memory": np.array(shape=(32, 64, 64), dtype=np.float32),
    "map_visible": np.array(shape=(32, 64, 64), dtype=np.int32),
}

# 提取第i个环境的观测
states = [(pos[^0], pos[^1]) for pos in obs_batch["agent_pos"]]  # 32个状态
```

三、动作空间

采用离散动作空间，共4个方向：

```
0: UP    (向上, +y方向)
1: DOWN  (向下, -y方向)
2: LEFT  (向左, -x方向)
3: RIGHT (向右, +x方向)
```

四、奖励机制与奖励塑形（Reward Shaping）

1. Reward Shaping配置

环境支持通过`options`参数自定义5个奖励分量：

```
# 默认配置
default_reward_shaping = {
    "per_step_reward": 0.0,       # 每步基础奖励
    "blocked_reward": 0.0,        # 碰撞障碍物/边界的惩罚
    "treasure_reward": 100.0,     # 收集单个宝箱的奖励
    "truncated_reward": 0.0,      # 超时结束的惩罚
    "terminated_reward": 150.0,   # 成功到达终点的奖励
}

# 你可以自定义配置来优化强化学习的路径
custom_conf = {
    "per_step_reward": 0.0,
    "blocked_reward": 0.0,
    "treasure_reward": 80.0,
    "truncated_reward": 0.0,
    "terminated_reward": 180.0,
}

env = GorgeWalkEnv(render_mode=None, options={"usr_conf": custom_conf})
```

2. 即时Reward计算逻辑

每步`env.step(action)`返回的`reward`按以下规则累加：

```
step_reward = 0

# 1. 碰撞检测
if 撞到障碍物或越界:
    step_reward += blocked_reward
else:
    更新agent位置

# 2. 移动惩罚
step_reward += per_step_reward

# 3. 宝箱奖励
if 当前位置有未收集的宝箱:
    step_reward += treasure_reward
    treasure_score += 100  # 用于total_score计算

# 4. 终止奖励
if 到达终点:
    step_reward += terminated_reward

if 超过最大步数:
    step_reward += truncated_reward
```

3. Total Score计算

`total_score`是任务的最终评估指标，计算规则在环境介绍处已经提及，下面是具体函数：

```
if terminated:  # 成功到达终点
    total_score = 150 + (max_step - current_step) × 1.0 + treasure_score
    # 150: 终点基础分
    # (max_step - current_step): 步数奖励，鼓励更快完成
    # treasure_score: 收集宝箱得分（每个100分）

elif truncated:  # 超时失败
    total_score = 0

else:  # 进行中
    total_score = treasure_score
```

注意：`step_reward`（即时奖励）和`total_score`（总得分）是两个独立的评估指标，前者往往用于训练引导，后者用于作为最终评估依据。

五、信息字典（Info）

如果你觉得observation中的信息以及基于这些信息处理的编码依然不够满足你的开发，info中还有一些额外信息，**你也可以拿来作为你的决策依据。**

1. 单环境Info结构

`env.step()`返回的`info`字典包含以下内容：

```
info = {
    "result_code": 0,
    "result_message": "Running" | "Success" | "Failure",

    "game_info": {
        "score": float,              # 本步即时reward
        "total_score": float,        # 当前总得分
        "step_no": int,              # 当前步数
        "pos_x": int,                # X坐标
        "pos_y": int,                # Y坐标
        "legal_act": [1,0,1,1],      # 4个动作的合法性（1=合法，0=非法）
        "treasure_count": int,       # 已收集宝箱数量
        "treasure_score": int,       # 宝箱累计得分
        "treasure_status": [0,1,1,0,...],  # 宝箱状态列表
    },

    "score_info": {
        "score": float,              # 同game_info["score"]
        "total_score": float         # 同game_info["total_score"]
    }
}
```

2. 异步并行环境Info格式

`AsyncVectorEnv`返回的`infos`是一个长度为N的列表：

```
next_obs_batch, rewards, terminated, truncated, infos = vec_env.step(actions)

# infos是list，长度为NUM_ENVS
print(len(infos))  # 32

# 访问第i个环境的info
final_score = infos[i]['game_info']['total_score']
```

注意：你可以参考示例代码来查看环境基本的利用方法以及他的函数的返回值。你也可以通过reset一次或者随机执行一次action，通过print的方式来查看具体的返回值，