Analyse the project

---

1. game.py

- Class `Board`
  建立游戏棋盘
  - `__init__`: 玩家，高，宽，n_in_row，状态字典(位置:玩家)
  - `init_board`: 初始化棋盘
  - `move_to_location`: 移动位置转换成坐标
  - `location_to_move`: 坐标(二维坐标)转换成移动位置(1维整数)
  - `current_state`: 返回当前的棋盘状态，4 * height * width,最高维分别是当前下子棋手的棋子标记，对手的棋子标记，对手上一步的走子标记，棋子颜色标记(黑棋的话全部标记为1)
  - `do_move`: 棋盘走子，更新最新步，交换 `current_player`
  - `has_a_winner`: 判断是否终止棋局，返回是否和赢家
  - `game_end`: 考虑平手的棋局最终判定，返回终止结果和赢家信息
  - `get_current_player`: 返回当前的走子方
- Class `Game`
  创建游戏，拥有 `Board` 类并负责在其上下棋，收集数据等等
  - `__init__`: 初始化，为游戏配备 `Board` 类
  - `graphic`: 将棋局信息打印在终端上
  - `start_play`: 玩家走子
  - `start_self_play`: 游戏自对弈，保存`(state,p,z)`数据作为训练集，如果游戏结束，负责重置 MCTS 搜索树

---

2. human_play.py

- Class `Human`
  人类玩家创建，人机对战的基础
  - `__init__`: 只需要初始化 `player` 类数据属性，将 `game.py/Board` 中的玩家数据和人类玩家之间绑定起来
  - `set_player_ind`: 在 `game.py/Game` 中调用将对应的玩家编号赋值给 `Human`
  - `get_action`: 键盘接收人类文件走子，对于错误的走子方案输入进行容错处理直至返回正确的结果传递给 `game.py/Game` 类处理
- Function `run`
  人机对战的运行函数，读取模型参数初始化 **策略价值网络** 和 MCTS 玩家，初始化游戏和对应的棋盘，将 `Human`, `mcts_player` 加入到游戏中运行 `game.py/Game/start_play` 函数开始人机对战

---

3. policy_value_net_tensorflow.py

- Class `PolicyValueNet`
  - `__init__`: 初始化网络结构，初始化所有变量参数
  - `policy_value`: 运行网络，返回走子策略和局面评估
  - `policy_value_fn`: 构建 batch ,调用 `policy_value` 生成走子策略和局面评估，实际下棋的时候调用该函数

- `train_step`: 输入 (局面，数据集中的走子策略，数据集中的价值数据，学习率)，执行一个训练步
- `save_model`: 保存模型参数
- `restore_model`: 加载模型参数

---

4. mcts_alphaZero.py

- Function `softmax`
  计算输入向量的 `softmax` 结果
- Class `TreeNode`
  - `__init__`: 初始化节点,(父节点，子节点字典，访问次数，Q,U,P)
  - `expand`: 扩展子节点并初始化各各子节点的先验概率
  - `select`: 根据 `Q+U` 的最大值选取对应的模拟走子方案，这里的计算的时候是计算子节点的值
  - `update`: 更新当前节点的数据
  - `update_recursive`: 调用 `update` 更新自身节点并反向递归更新祖先节点
  - `get_value`: 计算 `Q+U` 的值，被 `select` 函数调用，反向传播 `v` 值的时候，负值传播到父节点
  - `is_leaf`: 判断是否是子节点
  - `is_root`: 判断是否是当前的树根节点
- Class `MCTS`
  - `__init__`: 初始化树根，初始化 **策略价值网络函数** ，初始化超参数和模拟次数`_n_playout`
  - `_playout`: 执行一轮模拟
  - `get_move_probs`: 模拟`_n_playout`轮并返回当前的树根节点的每个子节点和对应的模拟次数构建的选择概率 `softmax`
  - `update_with_move`: 真实落子，如果落子是当前的根节点的子节点，复用搜索树，如果落子是 `-1` 重构搜索树
- Class `MCTSPlayer`
  - `__init__`: 初始化构建 `MCTS` 搜索树并决定改 `AI` 玩家目前是处在自对弈状态还是人机对弈状态
  - `set_player_ind`: 设置玩家编码
  - `reset_player`: 重构搜索树
  - `get_action`: 走子，如果是自对弈模式引入狄利克雷噪声提高采样落子多样性，否则人机对弈直接采样落子并且重构搜索树

---

5. train.py

- Class `TrainPipeline`
  - `__init__`: 初始化训练流程类 `TrainPieline` 的必要参数，其中重要的参数有棋面相关参数，`game` 类，学习率以及 KL 散度修正的动态学习速度，温度超参数，模拟次数，超参数 `cput`，训练数据队列(10000)，`mini_bacth` 大小定义(512)，迭代次数，最优胜率，纯蒙特卡洛的模拟次数，初始化构建对应的 **策略价值网络**，生成对应的 `MCTSPlayer`
  - `get_equi_data`: 逆时针转动和水平翻转增强数据集
  - `collect_selfplay_data`: 收集一局的自对弈数据并调用 `get_equi_data` 加强数据集，并将这些数据加入到训练数据队列中
  - `policy_update`: 使用一个 `mini_batch` 调用 `policy_value_net_tensorflow.py/train_step` 类函数更新网络参数，在项目中一个 `mini_batch` 被用来反向传播5次，并使用 KL 散度动态更新之后的学习速率，输出监视数据

- `policy_evaluate`: 和纯 MCTS 对弈计算胜率，如果绝对优势，提高 MCTS 的模拟次数之后重新迭代。本人认为这里可以使用一个相对来说更强的算法来代替这个 MCTS 对比程序可以保证网络的参数优化的更快。
- `run`: 启动训练管道，每一局自对弈结束之后输出监督信息，每50次迭代并且胜率提升迭代保存一次模型参数