Brain

+ uuid: string
-__memory: Memory
-__net: Poker_network

+ __init__(activation: string, env: HULH_Emulator, uuid: string, memory_size)
+ collect_samples(bet_history: list, hole_cards: list, community_cards: list, timestep: int, output: list)
+ train_net()
+ predict(hole: list, board: list, hist: list): ndarray
+ save()

DCFR

-__player: Brain
-__opponent: Brain
-__strategy: Brain
-__env: HULH_Emulator

+ __init__(env: HULH_Emulator)
-__save_tree()
+iterate(iterate: int, k_iter: int)
-__calc_strategy(hole: list, board: list, hist: list, traverser: Brain): list
-__traverse(state: State, events: dict, traverser: Brain, timestep: int): float

Poker_network

-__small_hidden: int
-__large_hidden: int
-__init_weights: RandomUniform
-__path_to_save: string
-__activation: string
-__learning_rate: float
-__batch_size: int
-__epochs: int
-__action_num: int
+ model: Model
-__cards_len: int

+ __init__(env: HULH_Emulator, activation: string, small_hidden: int, large_hidden: int, learning_rate: float)
+ loss_func(y_true: list, y_pred: list) : float
-__create_network(): Model
+ predict(hole: list, board: list, hist: list): ndarray
+ train_net(info_set: ndarray, output: ndarray, timesteps: list)
+ save_model(name: string)
+ clear_net()

HULH_Emulator

- __emulator : Emulator
+ player_uuid : string
+ opponent_uuid : string
+ SMALL_BLIND : int
+ BIG_BLIND : int
+ ROUNDS_LIMIT : int
+ ANTE : int
+ NUM_PLAYER : int
+ CARDS : list
+ STACK : int
+ ACTIONS_NUM : int
+ BET_HISTORY_LENGTH : int

+ initial_game()
- __convert_to_one_hot(cards: list): list
+ get_reward(state: State, events: dict, prev_turn: string): int
+ get_bet_history(events: dict): list
+ get_turn(state: State): string
+ is_terminal(state: State): bool
+ sample_action(state: State, prob: list): string
+get_community_cards(state: State): string
+ get_legal_actions(state: State): string
+ act(state: State, action: string): State, dict

Memory

-__info_sets: deque
-__timesteps: deque
-__outputs: deque
-__min_size: int

+ __init__(size: int, min_size: int)
+ append(hole: list, community: list, bet: list: timesteps: int, outputs: list)
+ get_storage(): deque, deque, deque
+ is_enough_samples(): bool