AIWolf Framework

1 gameinfo.py → 내 에이전트가 아는 데이터

```
    me: Agent "나"
    attack_vote_list: List[Vote] "늑대가 attack 할 에이전트를 선택하여 vote 한 리스트"
    latest_attack_vote_list: List[Vote] "가장 최신버전 attack_vote_list"
    attacked_agent: Optional[Agent] "attack 당한 에이전트 리스트"
    day: int "현재 날짜"
    divine_result: Optional[Judge] "divine 결과"
    executed_agent: Optional[Agent] "어젯밤 살해당한 에이전트 리스트 (vote 다수결로)"
    latest_executed_agent: Optional[Agent] "가장 마지막으로 살해당한 에이전트"
    existing_role_list: List[Role] "현재 게임에 존재하는 role 리스트"
    guarded_agent: Optional[Agent] "어젯밤 guarded 된 에이전트 리스트"
    last_dead_agent_list: List[Agent] "어젯밤 늑대의 attack 으로 죽임당한 에이전트 리스트"
    medium_result: Optional[Judge] "The result of the inquest."
    remain_talk_map: Dict[Agent, int] "남은 가능한 talk 횟수 (게임 설정에 따름)"
    remain_whisper_map: Dict[Agent, int] "남은 가능한 whisper 횟수"
    role_map: Dict[Agent, Role] "현재까지 알려진 에이전트의 role"
    status_map: Dict[Agent, Status] "각 에이전트의 생존여부"
    talk_list: List[Talk] "오늘의 talk 들 리스트"
    vote_list: List[Vote] "살해할 사람 vote 리스트"
    latest_vote_list: List[Vote] "가장 최신버전 vote_list"
    whisper_list: List[Whisper] "오늘의 whisper 들 리스트"
    def agent_list(self) -> List[Agent]: "존재하는 에이전트 리스트"
        return list(self.status_map.keys())
    def alive_agent_list(self) -> List[Agent]: "생존하고 있는 에이전트 리스트"
        return [i[0] for i in self.status_map.items() if i[1] == Status.ALIVE]
    def my_role(self) -> Role: "내 role."
        return self.role_map[self.me]
```

2 vote.py

```
class Vote:
       self, agent: Agent = AGENT_NONE, day: int = -1, target: Agent = AGENT_NONE
         agent(optional): vote 하는 에이전트.
         day(optional): vote 날짜
         target(optional): vote 당하는 에이전트.
```

3 utterance.py

```
class UtteranceType(Enum): "kind of utterance"
    TALK, WHISPER

class Utterance:
    self, day: int = -1, agent: Agent, idx: int = -1, text: str = "", turn: int = -1:
          day(opional): The date of the utterance.
          agent(optional): The agent that utters.
          idx(optional): The index number of the utterance.
          text(optional): The uttered text.
          turn(optional): The turn of the utterance.
class Talk(Utterance):
    self, day: int = -1, agent: Agent, idx: int = -1, text: str = "", turn: int = -1:
          idx(optional): The index number of the utterance.
          turn(optional): The turn of the utterance.
        super().__init__(day, agent, idx, text, turn)
```

```
class Whisper(Utterance):
    self, day: int = -1, agent: Agent, idx: int = -1, text: str = "", turn: int = -1:
        super().__init__(day, agent, idx, text, turn)
```

4 judge.py

```
class Judge: "human 인지 werewolf 인지 judgement"
    self, agent: Agent, day: int = -1, target: Agent, result: Species = Species.UNC:
          agent(optional): The agent that judged.
          day(optional): The date of the judgement.
          target(optional): The judged agent.
          result(optional): The result of the judgement.
```

5 agent.py

```
class Agent:
    _agent_map: ClassVar[Dict[int, Agent]] = {}
    _agent_pattern: ClassVar[Pattern[str]] = re.compile(r"(Agent\[(\d+)\]|ANY)")
    _agent_idx: int
    def __init__(self, idx: int) -> None: idx: The index number of the Agent.
        self._agent_idx = idx
    def agent_idx(self) -> int: "The index number of this Agent."
        return self._agent_idx

class Role(Enum):
    UNC, BODYGUARD, MEDIUM, POSSESSED, SEER, VILLAGER, WEREWOLF, ANY

class Species(Enum):
    UNC, HUMAN, WEREWOLF, ANY

class Status(Enum):
    UNC, ALIVE, DEAD
```

6 constant.py

```
class Constant: "Defines some constants."
    AGENT_NONE: Final[Agent] = Agent(0) ”No one”
    AGENT_UNSPEC: Final[Agent] = AGENT_NONE "Agent that means no agent specified"
    AGENT_ANY: Final[Agent] = Agent(0xff) "아무나"
```

7 content.py

```
class Content: "Content class expressing the content of an utterance."
    def __init__(self, builder: ContentBuilder) -> None:

class Topic(Enum):
    DUMMY, ESTIMATE, COMINGOUT, DIVINATION, DIVINED – Divination, Report of a divination
    IDENTIFIED = "IDENTIFIED" - Report of an identification
    GUARD, GUARDED, VOTE, VOTED, ATTACK, ATTACKED, AGREE, DISAGREE, Over, Skip, OPERATOR

class Operator(Enum):
    NOP = "NOP" – no operation
    REQUEST, INQUIRE = "INQUIRE" - inquiry, BECAUSE = "BECAUSE" – reason, DAY = "DAY" - 날짜
    AND, OR, XOR, NOT

class ContentBuilder:
    _subject:Agent, _target:Agent, _topic:Topic, _role:Role, _result:Species
    _utterance:Utterance, _operator: Operator, _content_list:List[Content], _day:int
```

```
class AgreeContentBuilder(ContentBuilder):
    self, utterance_type: UtteranceType, day: int, idx: int, *, subject: Agent:
            utterance_type: The type of the utterance.
            day: The date of the utterance.
            idx: The index number of the utterance.
            subject(optional): The agent that agrees. Defaults to AGENT_UNSPEC.
class DisagreeContentBuilder(AgreeContentBuilder):
    self, utterance_type: UtteranceType, day: int, idx: int, *, subject: Agent:

class AttackContentBuilder(ContentBuilder): − attack 의도 표현
    self, target: Agent, *, subject: Agent:
class AttackedContentBuilder(AttackContentBuilder): − report of attack
    self, target: Agent, *, subject: Agent:
            subject(optional): report 하는 당사자

class DivinationContentBuilder(AttackContentBuilder): "expressing a divination"
    self, target: Agent, *, subject: Agent = AGENT_UNSPEC:
            subject(optional): divination 하는 당사자.
class DivinedResultContentBuilder(ContentBuilder): "report of a divination"
    self, target: Agent, result: Species, *, subject: Agent:
            target: The agent that was an object of the divination.
            subject(optional): divination 한 당사자

class GuardContentBuilder(AttackContentBuilder): "expressing a guard"
    self, target: Agent, *, subject: Agent:
            target: The agent to be guarded.
            subject(optional): guard 하는 당사자
class GuardedAgentContentBuilder(AttackContentBuilder): "report of a guard."
    self, target: Agent, *, subject: Agent:
            target: The agent that was guarded.
            subject(optional): guard 한 당사자

class VoteContentBuilder(AttackContentBuilder): "expressing a vote"
    self, target: Agent, *, subject: Agent:
            subject(optional): The agent that votes.
class VotedContentBuilder(AttackContentBuilder):
    self, target: Agent, *, subject: Agent
            subject(optional): The agent that voted.

class ComingoutContentBuilder(ContentBuilder):
    self, target: Agent, role: Role, *, subject: Agent
            target: The agent that is an object of the comingout.
            subject(optional): comingout 하는 당사자
class EstimateContentBuilder(ComingoutContentBuilder):
    self, target: Agent, role: Role, *, subject: Agent
            target: The agent that is an object of the estimation.
            subject(optional): The agent that estimates.
class IdentContentBuilder(DivinedResultContentBuilder): "report of an identification"
    self, target: Agent, result: Species, *, subject: Agent
          target: The agent that was an object of the identification.
          result: The species of the agent revealed as a result of the identification.
          subject(optional): The agent that did the identification.
class RequestContentBuilder(ContentBuilder): "expressing a request"
    self, target: Agent, action: Content, *, subject: Agent
            target: The agent that is an object of the request.
            action: The requested action.
            subject(optional): The agent that requests. Defaults to AGENT_UNSPEC.
class InquiryContentBuilder(RequestContentBuilder): "expressing an inquiry"
    self, target: Agent, action: Content, *, subject: Agent
            target: The agent that reveives the inquiry.
            action: The matter inquired.
            subject(optional): The agent that makes the inquiry.
class BecauseContentBuilder(ContentBuilder):
    self, reason: Content, action: Content, *, subject: Agent:
            reason: The reason for the action.
            action: The action based on the reason.
            subject(optional): The agent that expresses the action and its reason.

class AndContentBuilder(ContentBuilder):
    self, contents: List[Content], *, subject: Agent
            contents: The series of the conjuncts.
            subject(optional): The agent that expresses the conjunctie clause.
class OrContentBuilder(AndContentBuilder):
    self, contents: List[Content], *, subject: Agent
            contents: The series of the disjuncts.
            subject(optional): The agent that expresses the disjunctive clause.
class XorContentBuilder(BecauseContentBuilder):
    self, disjunct1: Content, disjunct2: Content, *, subject: Agent
            disjunct1: The first disjunct.
            disjunct2: The second disjunct.
            subject(op): The agent that expresses the exclusive disjunctive clause.
class NotContentBuilder(ContentBuilder): - 부정하기
    self, content: Content, *, subject: Agent:
            content: The content to be negated.
            subject(optional): The agent that expresses the negation.
class DayContentBuilder(ContentBuilder): "adding a date to the Content"
    self, day: int, content: Content, *, subject: Agent:
            day: The date of the Content.
            content: The content to which the date is added.
            subject(optional): The agent that adds the date to the Content.
class SkipContentBuilder(ContentBuilder): - no variable
class OverContentBuilder(ContentBuilder): - no variable
class EmptyContentBuilder(ContentBuilder): - no variable
       self._topic = Topic.DUMMY
```

8 player.py

```
class AbstractPlayer(ABC): "defines the functions every player agents must have"
    def attack(self) -> Agent: Return the agent this werewolf wants to attack.
        The agent that does not exist means not wanting to attack any other.
    def day_start(self) -> None: Called when the day starts.
    def divine(self) -> Agent: Return the agent this seer wants to divine.
    def finish(self) -> None: Called when the game finishes
    def get_name(self) -> str: Return this player's name.
        return type(self).__name__
    def guard(self) -> Agent: Return the agent this bodyguard wants to guard.
        The agent that does not exist means no guard.
    def initialize(self, game_info: GameInfo, game_setting: GameSetting) -> None:
        Called when the game starts
    def talk(self) -> Content: Return this player's talk.
    def update(self, game_info: GameInfo) -> None: Called when the game information is updated
    def vote(self) -> Agent: Return the agent this player wants to exclude from this game.
        Returning the agent that does not exist results in ramdom vote.
    def whisper(self) -> Content:
```