BotAi

La classe `BotAi` fournit une interface haut niveau pour interagir avec le serveur de combat de bots. Il permet à un client de s'inscrire avec un bot, de lire les messages entrants du serveur et de réaliser des actions telles que se déplacer, tourner et tirer.

Propriétés de la classe

- `bot_id` : une propriété en lecture seule qui retourne l'identificateur du bot attribué par le serveur.

Méthodes de la classe

- `__init__(self, bot_name: str, team_id: str)` : Initialise l'instance de `BotAi` avec le `bot_name` et le `team_id` donnés.
- `__enter__(self)` : Permet l'utilisation de la déclaration `with` pour s'assurer que la méthode `close` est appelée.
- `__exit__(self, exc_type, exc_val, exc_tb)` : Appelé lorsqu'une déclaration `with` est quittée.
- `close(self)` : Ferme tous les threads ouverts.
- `enroll(self, bot_id: str = str()) -> str` : Inscrit ou réinscrit un bot sur le serveur. Donner un identificateur de bot existant vous permet de connecter ce bot sans en ajouter un nouveau au jeu. Retourne l'identificateur du bot attribué par le serveur.
- `read_scanner(self) -> dict` : Lit et retire un élément de la file d'attente du scanner. Retourne un dictionnaire contenant les données. **Exemples**
    - Un arbre et un bot sont détectés:


```
{
    "msg_type": "object_detection",
    "source": "scanner",
    "data": [
        {
            "from": 26.5,
            "to": 31,
            "object_type": "tree",
            "name": "Tree",
            "distance": 6.844473640068633
        },
        {
            "from": 34,
            "to": 39.5,
            "object_type": "bot",
            "name": "MyBot01",
            "distance": 5.777348380771669
        }
```

```
    ]
}
```

- `read_game_message(self) -> dict` : Lit et retire un élément de la file d'attente de messages de jeu. Retourne un dictionnaire contenant les données. **Exemples**
    - État de la partie:

```
{
    "msg_type": "game_status",
    "source": "server",
    "data": {
        "value": false
    }
}
```

    - Points de vies restants au bot:

```
{
    "msg_type": "health_status",
    "source": "bot",
    "data": {
        "value": 95
    }
}
```

    - Si le bot a été assomé (déplacements impossibles):

```
{
    "msg_type": "stunning_status",
    "source": "bot",
    "data": {
        "value": true
    }
}
```

    - Si le bot a commencé ou arrêté de se déplacer:

```
{
    "msg_type": "moving_status",
    "source": "bot",
    "data": {
        "value": false
    }
}
```

- Si le bot a commencé ou arrêté de tourner:

```
{
    "msg_type": "turning_status",
    "source": "bot",
    "data": {
        "value": false
    }
}
```

- Si l'arme du bot peut tirer ou non (rechargement):'

```
{
    "msg_type": "weapon_can_shoot",
    "source": "bot",
    "data": {
        "value": false
    }
}
```

- `move(self, state: bool)` : Commence ou arrête de faire avancer le bot. `state` peut être soit `True` ou `False`.
- `turn(self, direction: str)` : Commence ou arrête de tourner le bot dans une direction. `direction` peut être `"left"`, `"right"` ou `"stop"`.
- `shoot(self, angle: float)` : Tire à l'angle souhaité, spécifié en degrés.

Exceptions

- `RestException` : Levée en cas d'erreur lors d'un appel d'API REST.

Exemple d'utilisation

```
import logging
from time import sleep
from threading import Event, Thread
from queue import SimpleQueue
from battlebotslib.BotAi import BotAi


# Game information
G_GAME_IS_STARTED = False

# Bot information
G_BOT_HEALTH: int = 999     # Depends on the bot type, we need to read this
```

```
information from the game messages
G_BOT_IS_MOVING: bool = False
G_BOT_IS_TURNING: bool = False
G_BOT_TURN_DIRECTION: str = str()
G_BOT_IS_STUNNED: bool = False
G_WEAPON_CAN_SHOOT: bool = True

# Will be used to store all the objects to shoot at
G_BOT_TARGETS_QUEUE: SimpleQueue = SimpleQueue()


def thread_read_scanner_queue(e: Event, bot_ai: BotAi):
    """
    Thread continuously reading messages from the bot scanner queue.
    """
    while not e.is_set():
        scanner_message = bot_ai.read_scanner()
        logging.debug(f"[SCANNER] {scanner_message}")
        handle_scanner_message(scanner_message)


def thread_read_game_queue(e: Event, bot_ai: BotAi):
    """
    Thread continuously reading messages from the game queue.
    """
    while not e.is_set():
        game_message = bot_ai.read_game_message()
        logging.debug(f"[GAME] {game_message}")
        handle_game_message(game_message)


def handle_scanner_message(message: dict):
    """
    Handle a new scanner message.
    """
    try:
        if message['msg_type'] == "object_detection":
            # Browsing detected objects
            for detected_object in message['data']:
                is_valid_target = False
                target = None
                match detected_object['object_type']:
                    # We want to shoot at trees and bots
                    case "tree":
                        is_valid_target = True
                        target = detected_object
                    case "bot":
                        is_valid_target = True
                        target = detected_object
                    # We cannot walk on water
                    case "tile":
                        if detected_object["name"].lower() == "water":
                            logging.debug("WATER WATER WATER!!!")
                    case _:
```

```
                        pass

                if is_valid_target:
                    target_angle = (target['from'] + target['to']) / 2
                    logging.info(f"[SCANNER] {target['name']} detected at a
distance of "
                                 f"{target['distance']} ({target_angle}°)")
                    G_BOT_TARGETS_QUEUE.put(target_angle)
        else:
            logging.error(f"Unknown scanner message: {message}")
    except:
        logging.error(f"Bad scanner message format: {message}")


def handle_game_message(message: dict):
    """
    Handle a new game message.
    """
    try:
        match message['msg_type']:
            case "health_status":
                global G_BOT_HEALTH
                # Bot health update
                G_BOT_HEALTH = message['data']['value']
                show_bot_stats()
            case "game_status":
                global G_GAME_IS_STARTED
                # Game is running or stopped
                G_GAME_IS_STARTED = message['data']
            case "stunning_status":
                global G_BOT_IS_STUNNED
                # Bot is stunned or not
                G_BOT_IS_STUNNED = message['data']['value']
            case "moving_status":
                global G_BOT_IS_MOVING
                # Bot is moving or not
                G_BOT_IS_MOVING = message['data']['value']
            case "turning_status":
                global G_BOT_IS_TURNING
                global G_BOT_TURN_DIRECTION
                if message['data']['value'] == 'stop':
                    # Bot has been stopped
                    G_BOT_IS_TURNING = False
                else:
                    # Turn direction
                    G_BOT_IS_TURNING = True
                    G_BOT_TURN_DIRECTION = message['data']['value']
            case "weapon_can_shoot":
                global G_WEAPON_CAN_SHOOT
                G_WEAPON_CAN_SHOOT = message['data']['value']
            case _:
                logging.error(f"Unknown game message: {message}")
    except:
        logging.error(f"Bad game message format: {message}")
```

```
def show_bot_stats():
    logging.info(f"Health: {G_BOT_HEALTH}")


if __name__ == "__main__":
    # Logging
    logging.basicConfig(
        level=logging.DEBUG, datefmt='%d/%m/%Y %I:%M:%S', format='[%(levelname)s]
%(asctime)s - %(message)s'
    )

    # Creating a new Bot
    with BotAi(bot_name="MyBot", team_id="team-id-1") as bot:
        def stop():
            # Closing messages reading threads
            scanner_message_thread_event.set()
            game_message_thread_event.set()

        # Enrolling the new bot on the server
        bot_id = bot.enroll()

        # Bot scanner messages handler thread
        scanner_message_thread_event = Event()
        Thread(target=thread_read_scanner_queue, args=
(scanner_message_thread_event, bot)).start()

        # Game messages handler thread
        game_message_thread_event = Event()
        Thread(target=thread_read_game_queue, args=(game_message_thread_event,
bot)).start()

        try:
            # Waiting for the game to start
            while not G_GAME_IS_STARTED:
                sleep(0.1)

            # While the bot is alive and the game is running
            while G_BOT_HEALTH > 0 and G_GAME_IS_STARTED:
                # ################################################################
                #
                # AI logic goes here. Example:
                #
                # try:
                #     if not G_BOT_TARGETS_QUEUE.empty():
                #         bot.shoot(G_BOT_TARGETS_QUEUE.get(block=False))
                #     if not G_BOT_IS_MOVING:
                #         bot.move(True)
                #     if not G_BOT_IS_TURNING:
                #         bot.turn(random.choice(['left', 'right']))
                # except BotAi.RestException as ex:
                #     pass
                #
```

```
                # ##############################################################

            # Game has stopped or the bot is dead
            if G_BOT_HEALTH <= 0:
                logging.info("Bot is dead")
            elif not G_GAME_IS_STARTED:
                logging.info("Game has been stopped")

            stop()

        except KeyboardInterrupt:
            logging.info("Bot has been aborted")
            stop()
```