API Documentation

You can find the GitHub repository for the game engine here:
https://github.com/acm-cmu/awap-game-engine-2026-public

You can find the plain language guide here: Carnegie Cookoff Guide [OFFICIAL]

This is an exhaustive list of all the functions or methods that players are allowed to access. If you have any remaining questions about the functionality, please reach out to us at in person office hours!

Note: Please do not attempt to exploit the game engine by accessing the internal state. Attempting to do so may result in disqualification.

RobotController

These are all the member functions of RobotController. These are used to control your player and obtain information about the game state.

RobotController - Main Functionalities

State Access Functions

`get_turn() -> int`

Gets the current turn of the game.

`get_team() -> Team`

Returns the team you are on, returning either Team.RED or Team.BLUE.

`get_enemy_team() -> Team`

Returns your opponent's team, returning either Team.RED or Team.BLUE.

`get_map(team: Team) -> Map`

Returns the current map instance.

`get_orders(team: Team) -> List[Dict]`

Returns the list of orders, with each order represented by a dictionary.
Dictionary Items:

- **order_id:** id number for the order,
- **required:** the names of the foods required,
- **created_turn:** the turn the order appears,
- **expires_turn:** the turn the order will expire,
- **reward:** the reward for successfully submitting the order,
- **penalty:** penalty for letting the order expire,
- **claimed_by:** the id of the bot the order was claimed by,
- **completed_turn:** the turn that the order was submitted, or None,
- **is_active:** whether or not the order is currently active,

`get_team_bot_ids(team: Team) -> List[int]`

Returns the bot IDs of your team as a list.

`get_team_money(team: Team) -> int`

Returns the amount of money your team currently has.

`get_bot_state(bot_id:int) -> Optional[Dict]`

Returns the state of the given bot as a dictionary.
The dictionary contains:

- **bot_id:** the bot id
- **team:** the team name
- **x:** the bot's x-coordinate
- **y:** the bot's y-coordinate
- **team_money:** the amount of money the team has
- **holding:** the item the bot is currently holding
- **map_team:** the team associated to the map the bot is currently on

`get_tile(team:Team, x:int, y:int) -> Optional[Tile]`

Returns the tile at the given (x, y) coordinate.

Movement Functions

`can_move(bot_id:int, dx:int, dy:int) -> bool`

Returns whether the given bot can move to coordinates (dx, dy).

```
move(bot_id: int, dx: int, dy: int) -> bool
```

Moves the bot to coordinates (dx, dy); returns True if successful and False if not.

Inventory Functions

```
pickup(bot_id: int, target_x: Optional[int] = None, target_y:
Optional[int] = None) -> bool
```

Bot picks up an item from target location (target_x, target_y); returns True if successful and False if not. Can be used to pick up items from boxes.

```
place(bot_id: int, target_x: Optional[int] = None, target_y:
Optional[int] = None) -> bool
```

Bot places an item at target location (target_x, target_y); returns True if successful and False if not. Can be used to place items on pans and boxes.
Placement rules:

- can place anything in an empty box
- can only place food on a cooker with a pan on it

```
trash(bot_id: int, target_x: Optional[int] = None, target_y:
Optional[int] = None) -> bool
```

If coordinates (target_x, target_y) are the coordinates of the trash tile, the bot disposes of:

- the food item it is currently holding, or
- the food items in the pan or plate it is currently holding

And returns True. Otherwise, trash disposal fails and False is returned.

Shop Functions

```
can_buy(bot_id: int, item: Buyable, target_x: Optional[int] = None,
target_y: Optional[int] = None) -> bool
```

Checks if the bot can buy an item from the shop.
Requirements:

- target tile (target_x, target_y) is a shop tile
- bot is not carrying anything.

Returns True if possible, False otherwise.

```
buy(bot_id: int, item: Buyable, target_x: Optional[int] = None,
target_y: Optional[int] = None) -> bool
```

Buys an item from the shop.
Requirements:

- target tile (target_x, target_y) is a shop tile
- bot is not carrying anything.

Returns True if successful False otherwise.

Food Processing Functions

```
chop(bot_id: int, target_x: Optional[int] = None, target_y:
Optional[int] = None) -> bool
```

Chops food on a counter; returns True if successful, False if not.

```
can_start_cook(bot_id: int, target_x: Optional[int] = None, target_y:
Optional[int] = None) -> bool
```

Check if the bot can start cooking.
Requirements:

- target tile (target_x, target_y) is a cooker with an empty pan placed on it, and
- bot is carrying a cookable food item.

Returns True if possible, False otherwise.

```
start_cook(bot_id: int, target_x: Optional[int] = None, target_y:
Optional[int] = None) -> bool
```

Begins cooking at the cooker at tile (target_x, target_y).
Requirements:

- target tile (target_x, target_y) is a cooker with an empty pan placed on it, and
- bot is carrying a cookable food item.

If the food item has been previously cooked, the food will continue cooking at the beginning of the stage it was previously removed from the cooker at.
Returns True if cooking begins successfully, False otherwise.

```
take_from_pan(self, bot_id: int, target_x: Optional[int] = None,
target_y: Optional[int] = None) -> bool:
```

Removes food from the pan at tile (target_x, target_y).

Requirements:

- target tile (target_x, target_y) is a cooker with a non-empty pan placed on it, and
- bot is not carrying anything

Returns True if successful, False otherwise.

Plate Functions

```
take_clean_plate(self, bot_id: int, target_x: Optional[int] = None, target_y: Optional[int] = None) -> bool
```

Take a clean plate from the sink table at coordinates (target_x, target_y).
Requirements:

- sink table has available clean plates,
- target tile (target_x, target_y) is a sink table tile, and
- bot is not holding anything

Returns True if successful, False otherwise.

```
put_dirty_plate_in_sink(self, bot_id: int, target_x: Optional[int] = None, target_y: Optional[int] = None) -> bool
```

Place a dirty plate in the sink at coordinates (target_x, target_y).
Requirements:

- target tile (target_x, target_y) is a sink tile
- bot is holding a dirty plate

Returns True if successful, False otherwise.

```
wash_sink(self, bot_id: int, target_x: Optional[int] = None, target_y: Optional[int] = None) -> bool
```

Wash a dirty plate in the sink at coordinates (target_x, target_y).
Requirements:

- target tile (target_x, target_y) is a sink tile
- sink contains a dirty plate

Returns True if successful, False otherwise.

```
add_food_to_plate(self, bot_id: int, target_x: Optional[int] = None,
target_y: Optional[int] = None) -> bool
```

Either:

- add the food item at coordinates (target_x, target_y) to the plate the bot is holding
- add the food item the bot is holding to the plate at coordinates (target_x, target_y)

Requirements:

- target tile (target_x, target_y) contains food or a clean plate
- bot is holding food or a clean plate

Returns True if successful, False otherwise.

Submit Functions

```
can_submit(bot_id: int, target_x: Optional[int] = None, target_y:
Optional[int] = None) -> bool
```

Checks if the currently held plate can be submitted.

Requirements:

- target tile (target_x, target_y) is the submit station
- bot is holding a clean plate

Returns True if possible, False otherwise.

```
submit(bot_id: int, target_x: Optional[int] = None, target_y:
Optional[int] = None) -> bool
```

Submits the currently held plate.

Requirements:

- target tile (target_x, target_y) is the submit station
- bot is holding a clean plate

Returns True if possible, False otherwise. Warns if no order matches the submission.

Switch Functions

```
get_switch_info() -> Dict[str, Any]
```

Provides information about the switch, including turn, switch duration, and the switch status of both teams.

`can_switch_maps() -> bool`

Returns True if the user can switch into the enemy map, False otherwise.Can switch any time after turn 250 but can only switch once.

`switch_maps() -> bool`

Immediately teleports all bots on the user's team into the enemy map with non-interferring spawns. Does not consume a bot's move for that turn. Can only be called once per game per team.
Returns True if successful, false otherwise.

Item Information

`item_to_public_dict(it: Optional[Item]) -> Any`

Provides information about an item such as type, name, and ID.
Dictionary Items:

- **type:** "Food", "Plate", or "Pan"
    - Food Additional Items
        - **food_name:** name of the food
        - **food_id:** id number for the food type
        - **chopped:** whether or not the food is chopped
        - **cooked_stage:** 0 (uncooked), 1(cooked), or 2(burnt)
    - Plate Additional Items
        - **dirty:** whether or not the plate is dirty
        - **food:** list of dicts with above items
    - Pan Additional Items
        - **food:** a dictionary with information about the food item in the pan

Game Constants

`Map`

A class that details the environment

Convention: bottom-left is [0][0], top-right is [width-1][height-1]

The map is structured as follows:

```
                y == height -----
x == width    [[# # # # # # # #],
      |       [# # # # # # # #],
      |       [# # # # # # # #],
      |       [# # # # # # # #],
      v       [# # # # # # # #]]
```

Tile

A class containing the following fields:

- **tile_name:** name of the tile
- **tile_id:** id of the tile
- **is_walkable:** whether the tile is walkable
- **is_dangerous:** whether the tile is dangerous
- **is_placeable:** whether items can be placed on the tile
- **is_interactable:** whether the tile can be interacted with

Tile Types

There are 10 types of tiles:

- **Floor:** walkable floor tiles
- **Wall:** unwalkable wall tiles
- **Counter:** unwalkable tiles that items and ingredients can be placed on
- **Box:** unwalkable tiles that multiple ingredients of the same type can be placed in and removed from
- **Sink:** where dirty dishes are placed and washed
- **SinkTable:** where clean dishes appear after being washed
- **Cooker:** where ingredients are cooked
- **Trash:** where food is thrown away
- **Submit:** where orders are submitted
- **Shop:** where ingredients and items can be purchased

Ingredients

There are 5 different ingredients that can be processed and purchased.

Ingredient	ID	Choppable	Cookable	Cost
Egg	0	False	True	20
Onion	1	True	False	30

Meat	2	True	True	80
Noodles	3	False	False	40
Sauce	4	False	False	10

`Items`

There are 2 different items that can be purchased from the shop.

Item	**Cost**
Plate	2
Pan	4