Welcome to Mars Makeover!

Hello! We are so excited to see you at Algorithms with a Purpose 2023, hosted by ACM@CMU. This document contains the game rules, a reference manual for your code, and a list of the in-game constants.

If you have any questions feel free to ask one of our AWAP staff at Office Hours or on Discord.

>> Important Links

- Game Viewer and Mapmaker: https://awap.acmatcmu.com/view
- Dashboard: https://awap.acmatcmu.com/dashboard

Development & Submission Flow

>> Local Development

1. Clone game engine at https://github.com/ACM-CMU/awap-engine-2023-public
2. Write your bot in bot folder
3. Create a map using a mapmaker at https://awap.acmatcmu.com/view or use one of the 5 maps that we gave. Put the map in the map folder.
4. Specify players & maps in game_settings.json then run python3 run_game.py
5. Upload replay generated in replay folder to https://awap.acmatcmu.com/view to view gameplay!

>> Submission to Competition

1. Log into the dashboard with https://awap.acmatcmu.com/dashboard with the account username and password given
2. Upload your bot
    a. Request unranked scrimmages with other competitors or raffle bot
3. Review your match results
    a. Check match history
    b. Check leaderboard
    c. Download replay against other competitors

Game Overview

>> Background

Two competing countries would like to terraform a planet!

There are 3 main components/phases to the game:

1. Explore the map with explorer robots
2. Collect resources from metal deposits using miner robots
3. Terraform tiles using terraformer robots

Each player will use their fleet of robots to explore the planet, collect resources to expand their fleet, and terraform land to take over the planet. As the game state progresses, players can transform robots to satisfy their needs.

>> Objectives

The win condition is to terraform more tiles than your opponent by the end of the game.

>> Gameplay Setup

Each team begins with several initial terraformed tiles on the planet. The initial terraformed tiles have a terraform status of 5 (for blue, -5 for red team).

Visibility

At the start of the game, there is a fog of war obscuring most of the map. Tiles that are obscured by the fog of war are called hidden tiles. Tiles that are visible are called visible tiles. You can act with explorer robots to turn hidden tiles into visible tiles. Only your initial terraformed tiles and a radius of 1 tiles around them are initially visible, with all other tiles being hidden. You cannot see any information about tiles until they are visible.

Costs

Each turn, players can spend resources (metal) to spawn new robots (up to 1) at any terraformed tile. They can also move their robots based on direction. Some tiles have metal deposits on them. You can spawn miner robots to collect resources from metal deposits. Each robot mines metal pieces if it is on the same tile as a deposit. The amount of metal mined per turn depends on the specific deposit and ranges from 5 to 25.

Terraforming

Every non-deposit tile on the map has a terraformed status (integer from -10 to 10). Initial tiles have status of 0. You can act with terraformer robots to modify the terraformed status of tiles. Tiles with status of 0 are “neutral” and do not belong to either team. If the status is positive, it is your team’s tile. If the status is negative, it is considered the opponent’s tile. You can not terraform mining and impassable tiles.

Passability

Tiles are either passable or impassable. Impassable tiles are effectively out of play (robots can’t move onto or through them, you can’t terraform them, and there will not be metal deposits on them).

>> More about Robots

Robot Spawning

On the turn that a robot is spawned, it may not move or perform an action.

Robot Movement

Each turn, each robot may move to an adjacent tile (surrounding 8 tiles). A robot can move only once per turn. The function `move_robot` will apply the given direction to the robot.

Additionally, each robot is outfitted with an internal BFS algorithm. When computing a turn, you can select a “target destination” using the function `optimal_path`. The robot will calculate the shortest path to the target and return the dictionary of the optimal path’s directions mapped to their order number. A similar function `robot_to_base` exists for calculating the path to the nearest ally terraformed tile.

Note there is no cost to calling either `optimal_path` or `robot_to_base`.

Robot Collisions

Robots require a large space to operate and are easily damaged by obstacles. If robot A moves into a space occupied by robot B, robot A and robot B will be destroyed. Note that collisions apply regardless of team.

Robot Actions

Additionally, with one call to `robot_action`, each robot will take their action if possible. A robot can only take one action per turn.

1. Explorer
    a. Turns all of the 8 surrounding tiles visible, if they're not already
2. Miner
    a. Mines a tile if they are on a deposit (only works for mining tiles)
3. Terraformer
    a. Terraforms the tile they are on, increasing the terraform status by 1 (only works for terraform tiles)

Fatigue

Robots also have a limited battery. Each time they perform an action, they expend some energy. They can regain energy by standing on an ally-terraformed tile.

Transforming

In addition to spawning robots, you can also control them by changing their roles. These robots are advanced and can transform from one type to another, immediately changing into the selected type. Robots may move and perform the action of their new transformed type on the same turn as they transform. Note that there is a cost to transform.

Building

During each turn, a player can spend metal to build up to 1 new robot per terraformed tile. Note that there is a cost to spawn.

>> Robot Specs

Some details about the different types of robots:

Each robot has a battery level (represented by a number between 0 and a max of 120). When robots perform an "action", they reduce their battery level by a certain amount. If their battery level becomes less than their action cost, they are unable to perform an "action" and must return to an ally-terraformed tile to charge. When standing on an ally-terraformed tile, their battery level is restored by 30 per turn.

Note that moving is not considered an "action" and does not expend battery.

Robot Type	Action	Action cost
Explorer	Making a hidden tile visible	10
Miner	Collecting metal from a deposit	20
Terraformer	Terraforming a tile	20

>> Map Overview

The world is a square grid with width and height dimensions that may range from 16 to 48. Each map will have horizontal, vertical, or rotational symmetry. Tiles are either passable, impassable, or mineable.

More Game Details

>> Turn Structure

Each team begins the game with 200 metal pieces and some terraformed tiles.

Each game consists of 200 rounds with each team performing one turn per round.

At the start of each turn, a player is given information about their own game state (`game_state`). From the given game state, players may access the map from which they can see which tiles are hidden or visible. Among the visible tiles, they can also see which tiles are passable or impassable, information about metal deposits, and the terraforming status of each tile.

For each visible tile, players can also see the state of every robot on their visible tiles, including enemy robots.

A player will create a `play_turn` function that can use the game state to perform an action each turn. The player can move robots, transform robots, and perform robot actions in any order they want during each turn.

>> Illegal Actions

If any team attempts to take a useless action, an exception is raised. Actions that fall under this category include: taking an action without enough battery, moving to a hidden tile, moving to an impassable tile, terraforming a fully terraformed (status 10)

tile, mining on an unmineable tile, and exploring only tiles you have already explored (if, within the surround tiles, there is at least 1 unexplored tile then the action is valid).

>> Game End

The game ends after 200 rounds. We will then count up the number of non-zero terraformed tiles each team has on the map. The team with more terraformed tiles wins.

>> Tie Breaks

If a winner cannot be determined by number of terraformed tiles, it is determined in the following order:

1. Number of robots
2. Highest metal count
3. Highest remaining time bank

>> Time Limits

Each player's bot starts with a time bank of 1 second multiplied by the number of rounds with an extra padding of 10 seconds to your total round's time, and the time your turn takes to run is continuously subtracted from your time bank. If you deplete your time bank, the game will immediately end in a loss.

>> Allowed Packages

- Anything already imported anywhere in the game
- Python standard library
- Numpy
- Scipy

All packages not listed are not allowed - please let us know if there are packages you want to use and we'll consider them; we're flexible on this.

Player Reference Manual

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 contact reach out to us on #office-hours on discord.

Player-centric Classes

Students will create a child-class of Player that extends play_turn function:

```
class Player :
	# The team name
	team : Team

	# This contains code to run your turn. You will extend this function in your
file to define what your robot will do per turn.
	def play_turn(self, game_state: Gamestate) -> None
```

The child-class has to be named BotPlayer. Below is starter code you should use to make the child class:

```
from src.game_constants import RobotType, Direction, Team, TileState
from src.game_state import GameState, Game_Info
from src.player import Player
From src.map import TileInfo, RobotInfo


class BotPlayer(Player):
	"""
	A child class that implements the play_turn method
	"""
	def __init__(self, team: Team):
		Player.__init__(self, team)

	def play_turn(self, game_state: GameState) -> None:
		game_state_info = game_state.get_info()
		return
```

Game object to reference

Players use the game_state objects to get info and modify the game world.

```
class GameState :
	# Returns a GameInfo data class describing the game world
	def get_info() -> GameInfo

	# Returns a list of list of TileInfo representing the map
	def get_map() -> list[list[TileInfo]]

	# Returns a list of list of strings representing the map
	   -  I: Impassable Tile
	   -  M: Mining Tile
```

```
        - 0: Terraformable Tile
        - 1 to 10: Blue Terraformed Tile
        - -1 to -10 Red Terraformed Tile
        - #: Fog of War (unseen) Tile
    def get_str_map() -> list[list[str]]

    # Returns a dictionary of your robots with their string names as the keys and RobotInfo objects as the values (within your fog of war)
    def get_ally_robots() -> dict

    # Returns a dictionary of your enemy's robots with their string names as the keys and the RobotInfo objects as the value (within your fog of war)
    def get_enemy_robots() -> dict

    # Transforms the robot (using the robot's name) into RobotType. Returns a RobotInfo containing the transformed robot's info or None if it wasn't successful
    # Note: Robot's can only transform if player has enough metal
    def transform_robot (robot_name : str, type : RobotType) -> RobotInfo : class

    # Returns true if a player can transform the inputted robot given the restrictions mentioned above, false otherwise
    def can_transform_robot (robot_name : str, type : RobotType) -> RobotInfo : class


    # Moves the robot (using the robot's name) one tile in the dir direction. Return True if the action succeeded, False otherwise
    # Note: Robot's can move once per turn
    def move_robot (robot_name : str, dir : Direction) -> bool

    # Returns true if the inputted robot can move in the inputted direction, false otherwise
    def can_move_robot (robot_name : str, dir : Direction) -> bool


    # Robot takes an action (using the robot's name. Returns True for valid action
    # Note: Robot's can take actions till they run out of battery
    def robot_action (robot_name : str) -> bool

    # Returns true if the inputted robot can take an action on its current tile, false otherwise
    def can_robot_action (robot_name : str) -> bool


    # Creates a robot of RobotType at (row, col) coordinates. Returns a RobotInfo object for the newly created robot, or None if robot couldn't be created
    # Note: Robot's can only spawn if player has enough metal, the team has a terraform tile at (row, col), and no other robot exists at the tile
```

```
        def spawn_robot (type : RobotType, row : int, col : int) -> RobotInfo : class

        # Returns true if a player can spawn a robot on the tile at location [row, col] given the restrictions mentioned above, false otherwise
        def can_spawn_robot (type : RobotType, row : int, col : int) -> bool

        # Returns a robot's direction (using the robot's name) and number of turns it takes to get to the nearest terraformed tile. Returns a direction if it is possible or None otherwise. Check collisions gives a path that avoids robot collisions.
        def robot_to_base (robot_name : str, checkCollisions=True) -> tuple[Direction,int]

        # Returns a direction and number of turns it takes to follow the optimal path from coordinates (start_row, start_col) to (end_row, end_col). Check collisions gives a path that avoids robot collisions.
        # Note: Optimal Path doesn't work for any (r,c) outside the fog of war
        def optimal_path (start_row : int, start_col : int, end_row : int, end_col : int, checkCollisions=True) -> tuple[Direction,int]

        # Returns RobotInfo of a robot found on coordinates (row, col) or None if the tile is clear of robots
        # Note: Class will return None if (row,col) is in fog of war
        def check_for_collision (row : int, col : int) -> RobotInfo

        # Returns the quantity of metal owned by the current team in this round
        def get_metal () -> int

        # Returns the cost of spawning a robot
        def get_spawn_cost () -> int

        # Returns the cost of transforming a robot
        def get_transform_cost () -> int

        # Returns the current team enum
        def get_team () -> enum

        # Returns the number of the current turn
        def get_turn () -> int

        # Returns the quantity left in the current team's time bank for this game
        def get_time_left () -> int
```

Info Files (Accessible)

Players get GameInfo/RobotInfo objects as knowledge about the game

```
class GameInfo :
	# A dictionary of your robots organized [name : str, RobotInfo : class]
	robots: dict[str, RobotInfo]

	# A dictionary of enemy robots organized [name : str, RobotInfo : class]
	enemy_robots: dict[str, RobotInfo]

	# A list of lists of TileInfo of the map:
	map: list(list[TileInfo])

	# An integer representing how much metal you currently have
	metal: int

	# Your team name
	team: enum

	# The cost of spawning a robot, in terms of metal
	robot_spawn_cost: int

	# The cost of transforming a robot, in terms of metal
	robot_transform_cost: int

	# The amount of time a player has left at the start of a turn
	time_left: int

	# The current turn
	turn: int

class RobotInfo:
	# Contains the battery for a robot
	battery : int

	# True if the robot has acted this turn, False otherwise
	acted : bool

	# True if the robot has moved this turn, False otherwise
	moved : bool

	# Contains the robot's action cost
	action_cost : int

	# The robot's row
	row : int

	# The robot's col
	col : int
```

```
        # The robot's team
        team : Team

        # The robot's name
        name : str

        # The robot's type
        type : RobotType

class TileInfo:
        # Contains the tile state
        state : TileState

        # Terraform status of the tile
        terraform : int

        # Mining status of the tile (amount mineable)
        mining : int
```

Enum Class Reference

Players will use the following enum classes for the respective classes.

```
class Team:
        # Red Team
        RED : enum

        # Blue Team
        BLUE : enum

class RobotType:
        # The variable for the terraformable tile
        TERRAFORMABLE : enum

        # The variable for the mining tile
        MINING : enum

        # The variable for the impassable tile
        IMPASSABLE : enum

        # The variable for a tile that's not visible or out-of-bounds
        ILLEGAL : enum

class Tilestate:
        # The variable for the miner type of the robots
        MINER : enum
```

```
        # The variable for the terraformer type of the robots
        TERRAFORMER : enum

        # The variable for the explorer type of the robots
        EXPLORER : enum

class Direction :
        # The direction to move up from a current tile
        UP : enum (value: (-1,0))

        # The direction to move up and right from a current tile
        UP_RIGHT : enum (value: (-1,1))

        # The direction to move right from a current tile
        RIGHT : enum (value: (0,1))

        # The direction to move down and right from a current tile
        DOWN_RIGHT : enum (value: (1,1))

        # The direction to move down from a current tile
        DOWN : enum (value: (1,0))

        # The direction to move down and left from a current tile
        DOWN_LEFT : enum (value: (1,-1))

        # The direction to move left from a current tile
        LEFT : enum (value: (0,-1))

        # The direction to move up and left from a current tile
        UP_LEFT : enum (value: (-1,-1))
```

General Game Information

```
# Map Coordinate System
Using (row, col), a map's coordinates are:
```

```
(0,0), (0,1), (0,2), (0,3), (0,4), (0,5)
(1,0), (1,1), (1,2), (1,3), (1,4), (1,5)
(2,0), (2,1), (2,2), (2,3), (2,4), (2,5)
(3,0), (3,1), (3,2), (3,3), (3,4), (3,5)
(4,0), (4,1), (4,2), (4,3), (4,4), (4,5)
(5,0), (5,1), (5,2), (5,3), (5,4), (5,5)

Left Column = 0
Right Column = n-1
Top Row = 0
Bottom Row = n-1

# Map Constants
Number of Turns = 200

# TIME CONSTANTS
Time Limit = (Number of Turns) + 10 sec

# Currency Constants
Initial Battery = 120
Initial Metal = 200
Metal Gained Per Turn = 10
Mining Tile Minimum Amount = 5
Mining Tile Maximum Amount = 25
Maximum Terraformable Amount = 10

# Robot Action Constants
Explorer Robot’s Action Cost = 10
Miner Robot’s Action Cost = 20
Terraformer Robot’s Action Cost = 20

# Other Robot Constants
Robot’s Battery Recovered Per Turn at a Terraform Tile = 30
Metal Cost to Spawn Robot = 50
Metal Cost to Transform Robot = 40
```