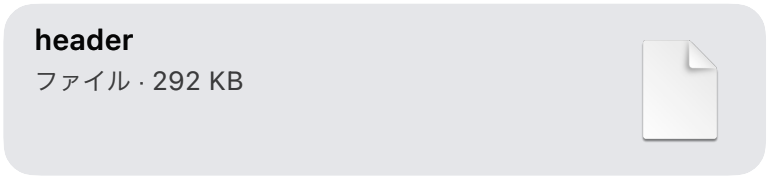
NeurIPS 2024 - Lux AI Season 3

Deep space exploration!



Overview

Welcome to Season 3! The goal of this competition is to create and/or train AI bots to play a novel multi-agent 1v1 game against other submitted agents. We hope to create a fun multi-agent competition for all in addition to advancing research into AI, imitation/reinforcement learning, and meta learning. Learn more about the competition specifics by reading below!

Start
Dec 10, 2024

Close
Mar 25, 2025

Merger & Entry

Description

linkkeyboard_arrow_up

Introduction

With the help 600+ space organizations, Mars has been terraformed

successfully. Colonies have been established successfully by multiple space organizations thanks to the rapidly growing lichen fields on the planet. The introduction of an atmosphere has enabled colonists to start thinking about the future, beyond Mars. Mysteriously, new deep-space telescopes launched from Mars revealed some ancient architectures floating beyond the solar system, hidden in a midst of asteroids and nebula gas. Perhaps they were relics of a previous sentient species?

Seeking to learn more about the secrets of the universe, new expeditions of ships were set out into deep space to explore these ancient relics and study them. What will they discover, which expedition will be remembered for the rest of history for unlocking the secrets of the relics?

The Lux AI Challenge is a competition where competitors design agents to tackle a multi-variable optimization, resource gathering, and allocation problem in a 1v1 scenario against other competitors. In addition to optimization, successful agents must be capable of analyzing their opponents and developing appropriate policies to get the upper hand.

All code can be found at our Github, make sure to give it a star while you are there!

Make sure to join our community discord to chat, strategize, and learn with other competitors! We will be posting announcements on the Kaggle Forums and on the discord.

Evaluation

linkkeyboard_arrow_up

Each day your team is able to submit up to 5 agents (bots) to the competition. Each submission will play Episodes (games) against other bots on the ladder that have a similar skill rating. Over time skill ratings will go up with wins or down with losses and evened out with ties. To reduce the number of bots playing and increase the number of episodes each team participates in, we only track the latest 2 submissions and use those for final submissions.

Every bot submitted will continue to play episodes until the end of the competition, with newer bots playing a much more frequent number of episodes. On the leaderboard only your best scoring bot will be shown, but you can track the progress of all of your submissions on your Submissions page.

Each Submission has an estimated Skill Rating which is modeled by a Gaussian $N(\mu,\sigma 2)$ where μ is the estimated skill and σ represents the uncertainty of that estimate which will decrease over time.

When you upload a Submission, we first play a Validation Episode

where that Submission plays against copies of itself to make sure it works properly. If the Episode fails, the Submission is marked as Error and you can download the agent logs to help figure out why. Otherwise, we initialize the Submission with μ0=600 and it joins the pool of All Submissions for ongoing evaluation.
We repeatedly run Episodes from the pool of All Submissions, and try to pick Submissions with similar ratings for fair matches. Newly submitted agents will be given an increased rate in the number of episodes run to give you faster feedback.

Ranking System

After an Episode finishes, we'll update the Rating estimate for all Submissions in that Episode. If one Submission won, we'll increase its μ and decrease its opponent's μ -- if the result was a draw, then we'll move the two μ values closer towards their mean. The updates will have magnitude relative to the deviation from the expected result based on the previous μ values, and also relative to each Submission's uncertainty σ. We also reduce the σ terms relative to the amount of information gained by the result. The score by which your bot wins or loses an Episode does not affect the skill rating updates.

Final Evaluation

At the submission deadline on March 24, 2025, additional submissions will be locked. From March 24, 2025 for approximately two weeks, we will continue to run games. At the conclusion of this period, the leaderboard is final.

Timeline

linkkeyboard_arrow_up

- **December 9, 2024** - Start Date.
- **March 3, 2025** - Entry Deadline. You must accept the competition rules before this date in order to compete.
- **March 3, 2025** - Team Merger Deadline. This is the last day participants may join or merge teams.
- **March 24, 2025** - Final Submission Deadline.
- **March 10, 2025 to March 24, 2025** - We will continue to run games. At the conclusion of this period, the leaderboard is final.

All deadlines are at 11:59 PM UTC on the corresponding day unless otherwise noted. *The competition organizers reserve the right to update the contest timeline if they deem it necessary.*

Prizes

linkkeyboard_arrow_up
Main Prizes
Monetary prizes are distributed to the top 8 competitors on the leaderboard at the end of the competition

- 1st Place: $10,000
- 2nd Place: $8,000
- 3rd Place: $7,000
- 4th-8th Place: $5,000

Sprint Prizes
At the end of each month of active competition*, the top 2 on the leaderboard will receive the Lux AI Challenge Sprint award for developing strong bots early on! This award can only be won once per team and if a team in the top 2 has already won the Sprint award, it will be given to the next highest ranking team who has not won the award. All teams are automatically entered into this competition.
In addition to the Sprint Award, teams will also be awarded competition t-shirts from competition organizers.
To pick the top 2 on the leaderboard, we will check the state of the leaderboard at 11:59PM UTC on the last day of each month.
Getting Started
linkkeyboard_arrow_up
All instructions in addition to the starter kits you need to start writing a bot and submit them to the competition are in this GitHub page. Make sure to also read the competition specs below to learn how this year's design works!
Lux AI Season 3 Specifications
linkkeyboard_arrow_up
For documentation on the API, see this document. To get started developing a bot, see our Github.
We are always looking for feedback and bug reports, if you find any issues with the code, specifications etc. please ping us on Discord or post a GitHub Issue. We will be fixing clear critical bugs as found. There will be a maximum of 2 balance patches, approximately one after 2 weeks of competition and one after 3 weeks of competition.
Background
With the help 600+ space organizations, Mars has been successfully terraformed successfully. Colonies have been established successfully by multiple space organizations thanks to the rapidly growing lichen fields on the planet. The introduction of an atmosphere has enabled colonists to start thinking about the future, beyond Mars. Mysteriously, new deep-space telescopes launched

from Mars revealed some ancient architectures floating beyond the solar system, hidden in a midst of asteroids and nebula gas. Perhaps they were relics of a previous sentient species?
Seeking to learn more about the secrets of the universe, new expeditions of ships were set out into deep space to explore these ancient relics and study them. What will they discover, which expedition will be remembered for the rest of history for unlocking the secrets of the relics?

Environment

In the Lux AI Challenge Season 3, two teams compete against each other on a 2D map in a best of 5 match sequence (called a game) with each match lasting 100 time steps. Both teams have a pool of units they can control to gain points around the map while also trying to prevent the other team from doing the same.
Unique to Season 3 is how various game mechanics and parameters are randomized at the start of each game and remain the same between matches in one game. Some mechanics/paramters include the map terrain/generation, how much units can see on the map, how might they be blocked by map features, etc. Each match is played with fog of war, where each team can only see what their own units can see, with everything else being hidden. Given that some mechanics are randomized between games, the specs will clearly document how they are randomized and what the possible values are. There is also a summary table of every game parameter that is randomized between games in the **Game Parameters** section below.
A core objective of this game is a balanced strategy of exploration and exploitation. It is recommended to explore more in the first match or two before leveraging gained knowledge about the map and opponent behavior to win the latter matches.

Map

The map is a randomly generated 2D grid of size 24x24. There are several core features that make up the map: Empty Tiles, Asteroid Tiles, Nebula Tiles, Energy Nodes, and Relic Nodes. Notably, in a game, the map is never regenerated completely between matches. Whatever is the state of the map at the end of one match is what is used for the next match.

Empty Tiles

These are empty tiles in space without anything special about them. Units and nodes can be placed/move onto these tiles.

Asteroid Tiles

Asteroid tiles are impassable tiles that block anything from moving/ spawning onto them. These tiles might move around over time during

the map in a symmetric fashion. Sometimes asteroid tiles might move on top of existing units. In the game the unit is not removed as a result of this and can still take actions and move around provided there is an non asteroid tile adjacent to it.

Nebula Tiles

Nebula tiles are passable tiles with a number of features. These tiles might move around over time during the map in a symmetric fashion.

Vision Reduction: Nebula tiles can reduce/block vision of units. Because of vision reduction it is even possible for a unit to be unable to see itself while still being able to move! See **Vision** section below for more details on how team vision is determined. All nebula tiles have the same vision reduction value called params.nebula_tile_vision_reduction which is randomized from 0 to 3.

Energy Reduction: Nebula tiles can reduce the energy of units that end their turn on them. All nebula tiles have the same energy reduction value called params.nebula_tile_energy_reduction.

Energy Nodes

Energy nodes are mysterious objects that emit energy fields which can be harvested by units. These nodes might move around over time during the map in a symmetric fashion. In code, what actually occurs in each game is energy nodes are randomly generated on the map symmetrically and a random function is generated for each node. Each energy node's function is a function of distance. The energy value of a tile on a map is determined to be the sum of the energy node functions applied to the distance between tile and each node.

Relic Nodes

Relic nodes are objects in space that enable ships to go near it to gain team points. These relic nodes however are ancient and thus fragmented. As a result, only certain tiles near the relic nodes when a friendly ship is on it will gain points. The tiles that yield points are always hidden and can only be discovered by trial and error by moving around the relic nodes. Relic node positions themselves can be observed if withins sensor range. The tiles around relic nodes can overlap with tiles of other relic nodes but will not yield extra points if that occurs and is treated as one tile.

In code, a random 5x5 configuration / mask centered on the relic node is generated indicating which tiles yield points and which don't. Multiple ships can stack on one tile but will only yield at most one point per tile. Note that ship stacking can be risky due to the **sapping action** (See **Sap Actions** section below).

Units

Units in the game are ships that can move one tile in 5 directions (center, up, right, down, left) and perform a ranged energy sapping action. Units can overlap with other friendly units if they move onto the same tile. Units have a energy property which determines whether they can perform actions and start with 100 energy and can have a max of 400 energy. Energy is recharged via the energy field of the map. They always spawn on one of the two corners of the map depending on which team they are on.

Note that nebula tiles and energy fields can modify the energy of a unit when it is on that tile. However they can never reduce the energy of a unit below 0, only opposing units can do that which will then remove the unit from the game to be respawned at a later timestep.

Unit IDs range from 0 to params.max_units - 1 for each team, and are recycled when units are spawned in if a previous one was removed.

Move Actions

All move actions except moving center
cost params.unit_move_cost energy to perform. Moving center is always free (a zero action). Attempting to move off the edge of the map results in no movement occuring but energy is still consumed. Units cannot move onto tiles with an impassible feature like an asteroid tile.

Sap Actions

The sap action lets a unit target a specific tile on the map within a range called params.unit_sap_range and reduces the energy of each opposition unit on the target tile by params.unit_sap_cost while also costing unit_sap_cost energy to use. Moreover, any opposition units on the 8 adjacent tiles to the target tile are also sapped and their energy is reduced by params.unit_sap_cost * params.unit_sap_dropoff_factor.

Sap actions are submitted to the game engine / environment as a delta x and delta y value relative to the unit's current position. The delta x and delta y value magnitudes must both be <= params.unit_sap_range, so the sap range is a square around the unit.

Generally sap actions are risky since a single miss means your ships lose energy while the opponent does not. The area of effect can mitigate this risk somewhat depending on game parameters. Sap actions can however prove very valuable when opposition ships are heavily stacked and get hit as sapping the stacked tile hits every ship on the tile.

Vision

A team's vision is the combined vision of all units on that team. Team vision is essentially a boolean mask / matrix over the 2D map indicating whether that tile's information is visible to the team. In this game, you can think of each unit having an "eye in the sky" sattelite that is capturing information about the units surroundings, but this sattelite has reduced accuracy the farther away the tile is from the unit.

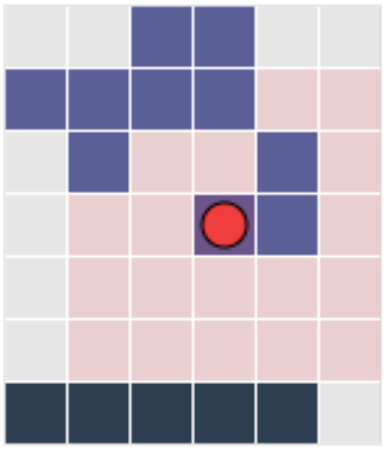
To determine which map tiles are visible to a team, we compute a vision power value for each tile on the map. For each unit on a team, we check each tile within the unit's sensor range and add 1 + params.unit_sensor_range - min(dx, dy) to the vision power map at tile (x+dx, y+dy) where (x,y) is the unit's position and (dx, dy) is the offset from the unit's position and abs(dx) <= params.unit_sensor_range and abs(dy) <= params.unit_sensor_range.

Nebula tiles have a vision reduction value of params.nebula_tile_vision_reduction. This number is reduced from every tile's vision power if that tile is a nebula tile.

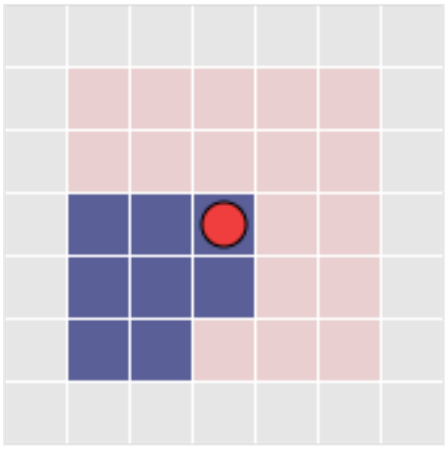
For example, naturally without any nebula tiles the vision power values look like below and create a square of visibility around the unit.

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	2	2	2	1	0
0	1	2	3	2	1	0
0	1	2	2	2	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

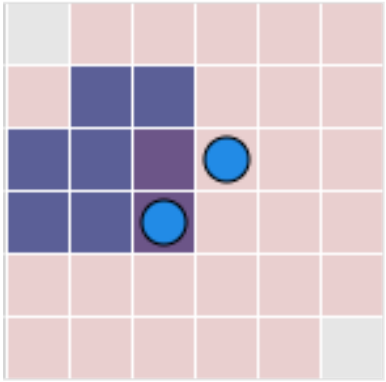
When a unit is near a nebula tile, it can't see details about some nebula tiles, but it can see tiles beyond nebula tiles. Here the unit has a sensor range of 2 and the nebula tile vision reduction value is 2. It can see itself since the vision power centered at the unit is 3, but it can't see other nebula tiles since they are too far or the nebula tile vision reduction reduces the vision power to 0 or less.



When a unit is inside a nebula tile, if the nebula vision reduction is powerful enough (here the nebula vision reduction is 3, unit sensor range is 2), the unit cannot even see itself or any other nebula tiles.



Unit vision can overlap and increase the vision power linearly, which can help handle the situations like above when you cannot see anything. Below the nebula vision reduction is 3 and the unit sensor range is 2, and now some of the nebula tiles are visible thanks to the overlapping vision of two units.



Collisions / Energy Void Fields

In close quarters, units can impact each other in two ways, via direct collisions or by being adjacent to each other and sapping energy via their energy void fields.

In the event of two or more units from opposing teams occupy the

same tile at the end of a turn, the team with the highest aggregate energy among its units on that tile survive, while the units of the opposing teams are removed from the game. If it is a tie, all units are removed from the game.
Furthermore, each unit generates an "energy void" field around itself that affects all cardinally (up, right, down left) adjacent opposition units. To determine how exactly each unit is affected by these energy void fields, we compute a 2D map for each team indicating the energy void strength at each tile. A unit contributes to tiles adjacent to itself a energy void strength equal to the total amount of energy the unit has at the start of the turn multiplied
by params.unit_energy_void_factor rounded down. After a energy void map is computed for each team, a unit's energy is reduced by the energy void strength of the tile it is on divided by the total number of units on that tile. Note that units removed due to collisions do not contribute to the energy void field.
The energy void fields generally encourage stacking units to better spread out energy sapped by energy void fields of opposition units.

Win Conditions

To win the game, the team must have won the most matches out of the 5 match sequence.
To win a match, the team must have gained more relic points than the other team at the end of the match. If the relic points scores are tied, then the match winner is decided by who has more total unit energy. If that is also tied then the winner is chosen at random.

Match Resolution Order

At each time step of a match, we run the following steps in order:

1. Move all units that have enough energy to move
2. Execute the sap actions of all units that have enough energy to do so
3. Resolve collisions and apply energy void fields
4. Update the energy of all units based on their position (energy fields and nebula tiles)
5. Spawn units for all teams. Remove units that have less than 0 energy.
6. Determine the team vision / sensor masks for all teams and mask out observations accordingly
7. Environment objects like asteroids/nebula tiles/energy nodes move around in space
8. Compute new team points

Note that each match runs for params.max_steps_in_match steps

and you take that many actions that affect the game. However, you will actually receive params.max_steps_in_match + 1 frames of observations since the very first frame will either be empty or the previous match's final observation (actions on these observations will not do anything).

Game Parameters

The full set of game parameters can be found here in the codebase.

Randomized Game Parameters / Map Generation

There are a number of randomized game paramteres which can modify and even disable/enable certain game mechanics. None of these game parameters are changed between matches in a game. The majority of these parameters are also not given to the teams themselves and must be discovered through exploration.

```
env_params_ranges = dict(
    map_type=[1],
    unit_move_cost=list(range(1, 6)), # list(range(x, y)) = [x, x+1,
x+2, ... , y-1]
    unit_sensor_range=list(range(2, 5)),
    nebula_tile_vision_reduction=list(range(0,4)),
    nebula_tile_energy_reduction=[0, 0, 10, 25],
    unit_sap_cost=list(range(30, 51)),
    unit_sap_range=list(range(3, 8)),
    unit_sap_dropoff_factor=[0.25, 0.5, 1],
    unit_energy_void_factor=[0.0625, 0.125, 0.25, 0.375],
    # map randomizations
    nebula_tile_drift_speed=[-0.05, -0.025, 0.025, 0.05],
    energy_node_drift_speed=[0.01, 0.02, 0.03, 0.04, 0.05],
    energy_node_drift_magnitude=list(range(3, 6))
)
```

These parameter ranges (and other parameters) are subject to change in the beta phase of this competition as we gather feedback and data.

Using the Visualizer

The visualizer will display the state of the environment at time step t out of some max number indicated in the page under the map. Actions taken at timestep t will affect the state of the game and be reflected in the next timestep t+1.

There are also a few important keyboard shortcuts that can be used to toggle various visualizations, namely

- s: Sensor range of all units
- e: The current energy field of the map
- r: The relic node tiles of the map

Events

linkkeyboard_arrow_up

We will live stream events on our Twitch. There will be sprint livestreams as well as a finals event which happens after the final evaluation period is over.
Make sure to follow our social media as well to be notified of future events and when we go live!

- Discord
- X/Twitter
- Twitch
- Youtube

Citation

linkkeyboard_arrow_up

Stone Tao, Akarsh Kumar, Bovard Doerschuk-Tiberi, Isabelle Pan, Addison Howard, and Hao Su. NeurIPS 2024 - Lux AI Season 3. https://kaggle.com/competitions/lux-ai-season-3, 2024. Kaggle.

Dataset Description

This is the folder for the Python kit. Please make sure to read the instructions as they are important regarding how you will write a bot and submit it to the competition.
For kits in other languages please see our Github repository

Files
6 files
Size
9.54 kB
Type
py, md
License
Apache 2.0
README.md(2.79 kB)

get_app
fullscreen

chevron_right

Lux AI Season 3 Python Kit

This is the folder for the Python kit. Please make sure to read the instructions as they are important regarding how you will write a bot and submit it to the competition. For those who need to know what python packages are available on the competition server, see this
Make sure to check our Discord for announcements if there are any breaking changes.
or the Kaggle forums

Requirements

You will need Python 3.9 or higher and NumPy installed (which should come with the dependencies you installed for the environment). To install the environment run
pip install --upgrade luxai-s3

Getting Started

To get started, download this folder from this repository.
Your core agent code will go into agent.py, and you can create and use more files to help you as well. You should leave main.py alone as that code enables your agent to compete against other agents locally and on Kaggle.
To quickly test run your agent, run
luxai-s3 main.py main.py --output=replay.html

This will run the agent.py code in the same folder as main.py and generate a replay file saved to replay.html that you can open and watch.

Developing

Now that you have the code up and running, you are ready to start programming and having some fun!
If you haven't read it already, take a look at the design specifications for the competition. This will go through the rules and objectives of the competition.
All of our kits follow a common API through which you can use to access various functions and properties that will help you develop your strategy and bot. The markdown version is here: https://github.com/Lux-AI-Challenge/Lux-Design-S3/blob/main/kits/README.md, which also describes the observation and action structure/spaces.

Submitting to Kaggle

Submissions need to be a .tar.gz bundle with main.py at the top level directory (not nested). To create a submission, create the .tar.gz with tar -czvf submission.tar.gz *. Upload this under the My Submissions tab and you should be good to go! Your submission will start with a scheduled game vs itself to ensure everything is working before being entered into the matchmaking pool against the rest of the leaderboard.

Data Explorer

9.54 kB

- arrow_right
  folder
  lux
- article
  README.md
- code
  agent.py
- code
  main.py

Summary

arrow_right

folder

6 files

get_app

Download All

DOWNLOAD DATA

navigate_next

minimize

kaggle competitions download -c lux-ai-season-3

content_copy
help

DOWNLOAD DATA

text_snippet

PPO (Stable-Baselines3)

Modifications in V8 Version

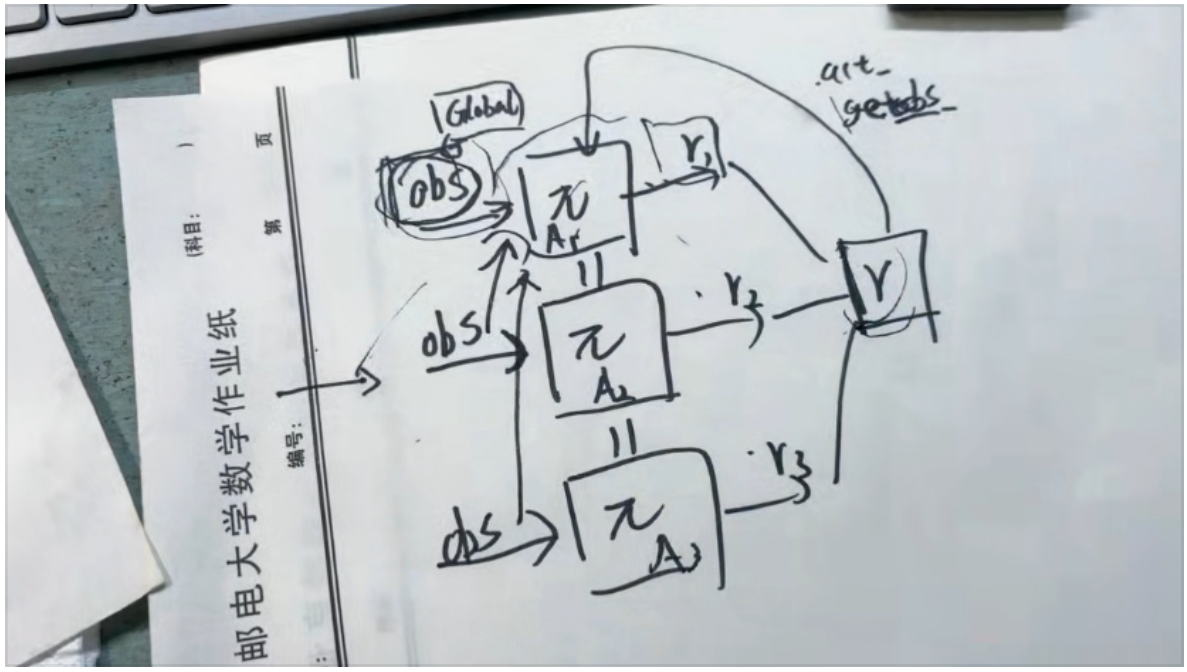
1. Simulate the Competition Data Transmission Format in the Training Environment:

```
// T is the number of teams (default is 2)
// N is the max number of units per team
// W, H are the width and height of the map
// R is the max number of relic nodes
{
  "obs": {
    "units": {
      "position": Array(T, N, 2),
      "energy": Array(T, N, 1)
    },
    // Indicates whether the unit exists and is visible to you.
units_mask[t][i] shows if team t's unit i can be seen and exists.
    "units_mask": Array(T, N),
    // Indicates whether the tile is visible to the unit for that team
    "sensor_mask": Array(W, H),
    "map_features": {
        // Amount of energy on the tile
        "energy": Array(W, H),
        // Type of the tile. 0 is empty, 1 is a nebula tile, 2 is asteroid
        "tile_type": Array(W, H)
    },
    // Indicates whether the relic node exists and is visible to you.
    "relic_nodes_mask": Array(R),
```

```
    // Position of the relic nodes.
    "relic_nodes": Array(R, 2),
    // Points scored by each team in the current match
    "team_points": Array(T),
    // Number of wins each team has in the current game/episode
    "team_wins": Array(T),
    // Number of steps taken in the current game/episode
    "steps": int,
    // Number of steps taken in the current match
    "match_steps": int
  },
  // Number of steps taken in the current game/episode
  "remainingOverageTime": int, // Total amount of time your bot can use whenever it exceeds 2s in a turn
  "player": str, // Your player id
  "info": {
    "env_cfg": dict // Some of the game's visible parameters
  }
}
```

2. Modify the Gradual Increase in the Number of Agents (According to Competition Rules) 3. Calculate the Field of View (Based on Different Agent IDs)

In the competition, the JSON returns the field of view observation for an entire team, so in order to calculate different fields of view for each unit and generate different predictions, the architecture is as follows:



4. Reward Function Optimization

1. Each unit calculates its `unit_reward` independently.

2. If a movement action causes the unit to move out of bounds or onto a target tile that is an Asteroid, the action is deemed invalid, and `unit_reward` is reduced by -0.2.

3. **Sap action:**

- Check whether the `relic_nodes_mask` in the unit's local observation contains a relic.
- If a relic is present, count the number of enemy units within the unit's 8-neighbor region:
  - If the count is **≥2**, the sap reward is **+1.0 × the number of enemy units**; otherwise, a **penalty of -2.0** is applied.
- If no relic is visible, a **penalty of -2.0** is also applied.

4. **Non-sap actions:**

- After a successful movement, check if the unit is located at a

  potential point

  configured for relic placement:

  - If it is the **first visit** to this potential point, `unit_reward` increases by **+2.0**, and it is marked as `visited`.
  - If the potential point has **not yet contributed to the team's score**, increase `self.score` by 1, add **+5.0** to `unit_reward`, and mark it as `team_points_space`.
  - If the unit is **already on a `team_points_space`**, it receives a **+5.0** reward each turn.

- If the unit is on an **energy node** (`energy == Global.MAX_ENERGY_PER_TILE`), `unit_reward` increases by **+0.2**.

- If the unit is on a **Nebula** (`tile_type == 1`), `unit_reward` decreases by **-0.2**.

- If the unit moves and overlaps with an enemy unit **while having higher energy than the enemy**, each enemy unit that meets this condition grants a **+1.0** reward.

5. **Global exploration reward:** Each newly discovered tile within the **combined vision** of all friendly units grants **+0.1** reward per tile.

6. At the **end of each step**, the final reward is calculated as **(point reward × 0.3) + (rule-based reward × 0.7)**.
5. Load the Original Model into the Steps Process for Adversarial Training (To Be Completed)

Existing Issues

The final training results do not align with the reward function design. This should be the final version of the notebook. It serves as a good starting framework (although its results still need optimization).

In [ ]:

```
! pip install stable-baselines3
```

In [ ]:

```
!mkdir agent && cp -r ../input/lux-ai-season-3/* agent/
import sys
sys.path.insert(1, 'agent')
```

base.py

game constants and some useful functions

In [ ]:

```
%%writefile agent/base.py

from enum import IntEnum


class Global:

    # Game related constants:

    SPACE_SIZE = 24
    MAX_UNITS = 16
    RELIC_REWARD_RANGE = 2
    MAX_STEPS_IN_MATCH = 100
    MAX_ENERGY_PER_TILE = 20
    MAX_RELIC_NODES = 6
    LAST_MATCH_STEP_WHEN_RELIC_CAN_APPEAR = 50
    LAST_MATCH_WHEN_RELIC_CAN_APPEAR = 2
```

```
    # We will find the exact value of these constants during the game
    UNIT_MOVE_COST = 1  # OPTIONS: list(range(1, 6))
    UNIT_SAP_COST = 30  # OPTIONS: list(range(30, 51))
    UNIT_SAP_RANGE = 3  # OPTIONS: list(range(3, 8))
    UNIT_SENSOR_RANGE = 2  # OPTIONS: [1, 2, 3, 4]
    OBSTACLE_MOVEMENT_PERIOD = 20  # OPTIONS: 6.67, 10, 20, 40
    OBSTACLE_MOVEMENT_DIRECTION = (0, 0)  # OPTIONS: [(1, -1), (-1, 1)]

    # We will NOT find the exact value of these constants during the game
    NEBULA_ENERGY_REDUCTION = 5  # OPTIONS: [0, 1, 2, 3, 5, 25]

    # Exploration flags:

    ALL_RELICS_FOUND = False
    ALL_REWARDS_FOUND = False
    OBSTACLE_MOVEMENT_PERIOD_FOUND = False
    OBSTACLE_MOVEMENT_DIRECTION_FOUND = False

    # Game logs:

    # REWARD_RESULTS: [{"nodes": Set[Node], "points": int}, ...]
    # A history of reward events, where each entry contains:
    # - "nodes": A set of nodes where our ships were located.
    # - "points": The number of points scored at that location.
    # This data will help identify which nodes yield points.
    REWARD_RESULTS = []

    # obstacles_movement_status: list of bool
    # A history log of obstacle (asteroids and nebulae) movement events.
    # - `True`: The ships' sensors detected a change in the obstacles' positions at this step.
    # - `False`: The sensors did not detect any changes.
    # This information will be used to determine the speed and direction of obstacle movement.
    OBSTACLES_MOVEMENT_STATUS = []
```

```
    # Others:

    # The energy on the unknown tiles will be used in the pathfinding
    HIDDEN_NODE_ENERGY = 0


SPACE_SIZE = Global.SPACE_SIZE


class NodeType(IntEnum):
    unknown = -1
    empty = 0
    nebula = 1
    asteroid = 2

    def __str__(self):
        return self.name

    def __repr__(self):
        return self.name


_DIRECTIONS = [
    (0, 0),  # center
    (0, -1),  # up
    (1, 0),  # right
    (0, 1),  #  down
    (-1, 0),  # left
    (0, 0),  # sap
]


class ActionType(IntEnum):
    center = 0
    up = 1
    right = 2
    down = 3
    left = 4
    sap = 5

    def __str__(self):
```

```
        return self.name

    def __repr__(self):
        return self.name

    @classmethod
    def from_coordinates(cls, current_position, next_position):
        dx = next_position[0] - current_position[0]
        dy = next_position[1] - current_position[1]

        if dx < 0:
            return ActionType.left
        elif dx > 0:
            return ActionType.right
        elif dy < 0:
            return ActionType.up
        elif dy > 0:
            return ActionType.down
        else:
            return ActionType.center

    def to_direction(self):
        return _DIRECTIONS[self]



def get_match_step(step: int) -> int:
    return step % (Global.MAX_STEPS_IN_MATCH + 1)


def get_match_number(step: int) -> int:
    return step // (Global.MAX_STEPS_IN_MATCH + 1)


# def warp_int(x):
#     if x >= SPACE_SIZE:
#         x -= SPACE_SIZE
#     elif x < 0:
#         x += SPACE_SIZE
#     return x
```

```
# def warp_point(x, y) -> tuple:
#     return warp_int(x), warp_int(y)


def get_opposite(x, y) -> tuple:
    # Returns the mirrored point across the diagonal
    return SPACE_SIZE - y - 1, SPACE_SIZE - x - 1


def is_upper_sector(x, y) -> bool:
    return SPACE_SIZE - x - 1 >= y


def is_lower_sector(x, y) -> bool:
    return SPACE_SIZE - x - 1 <= y


def is_team_sector(team_id, x, y) -> bool:
    return is_upper_sector(x, y) if team_id == 0 else is_lower_sector(x,
y)
```

ppo_game_env.py

In [ ]:

```
%%writefile agent/ppo_game_env.py
import sys
import gym
from gym import spaces
import numpy as np

# 导入 base 中的全局常量和辅助函数
from base import Global, ActionType, SPACE_SIZE, get_opposite

# 定义常量：队伍数、最大单位数、最大遗迹节点数
NUM_TEAMS = 2
MAX_UNITS = Global.MAX_UNITS
MAX_RELIC_NODES = Global.MAX_RELIC_NODES

class PPOGameEnv(gym.Env):
    """
    PPOGameEnv 模拟环境尽可能还原真实比赛环境，并满足以下要求：
```

1. 观察数据计算修改：
- 每个己方单位均有自己独立的 sensor mask（由 compute_unit_vision(unit) 计算），
并由 get_unit_obs(unit) 构造出符合固定格式的局部观察（字典形式）。
- 返回给代理的全局观察则采用所有己方单位 sensor mask 的联合（逻辑“或”），
保持比赛返回 obs 的固定格式。

2. 奖励函数优化：
根据动作更新环境状态，并返回 (observation, reward, done, info)。
修改后的奖励逻辑：
1. 每个 unit 单独计算 unit_reward。
2. 若移动动作导致超出地图或目标 tile 为 Asteroid，则判定为无效，unit_reward -0.2。
3. Sap 动作：
- 检查 unit 局部 obs 中 relic_nodes_mask 是否存在 relic；
- 如果存在，统计 unit 8 邻域内敌方单位数，若数目>=2，则 sap 奖励 = +1.0× 敌方单位数，否则扣 -2.0；
- 若无 relic 可见，则同样扣 -2.0。
4. 非 sap 动作：
- 成功移动后，检查该 unit 是否位于任一 relic 配置内的潜力点：
* 若首次访问该潜力点，unit_reward +2.0，并标记 visited；
* 如果该潜力点尚未兑现 team point，则增加 self.score 1，同时 unit_reward +3.0 并标记为 team_points_space；
* 如果已在 team_points_space 上，则每回合奖励 +3.0；
- 若 unit 位于能量节点（energy == Global.MAX_ENERGY_PER_TILE），unit_reward +0.2；
- 若 unit 位于 Nebula（tile_type==1），unit_reward -0.2；
- 如果 unit 移动后与敌方 unit 重合，且对方能量低于己方，则对每个满足条件的敌方 unit 奖励 +1.0。
5. 全局探索奖励：所有己方单位联合视野中新发现 tile，每个奖励 +0.1。
6. 每一step结束，奖励 point*0.5的奖励 + 规则*0.5的奖励

3. 敌方单位策略说明：
- 敌方单位在出生后不主动行动，其位置仅由环境每 20 步整体滚动（右移 1 格）改变，
属于被动对手。这样设计主要用于初期调试，后续可引入更主动的对抗策略。
"""

```
    def __init__(self):
        super(PPOGameEnv, self).__init__()

        # 修改动作空间：每个单位独立决策（动作取值范围为 0~5）
        self.action_space = spaces.MultiDiscrete([len(ActionType)] *
MAX_UNITS)

        # 观察空间保持不变
        self.observation_space = spaces.Dict({
            "units_position": spaces.Box(
                low=0,
                high=SPACE_SIZE - 1,
                shape=(NUM_TEAMS, MAX_UNITS, 2),
                dtype=np.int32
            ),
            "units_energy": spaces.Box(
                low=0,
                high=400,  # 单位能量上限 400
                shape=(NUM_TEAMS, MAX_UNITS, 1),
                dtype=np.int32
            ),
            "units_mask": spaces.Box(
                low=0,
                high=1,
                shape=(NUM_TEAMS, MAX_UNITS),
                dtype=np.int8
            ),
            "sensor_mask": spaces.Box(
                low=0,
                high=1,
                shape=(SPACE_SIZE, SPACE_SIZE),
                dtype=np.int8
            ),
            "map_features_tile_type": spaces.Box(
                low=-1,
                high=2,
                shape=(SPACE_SIZE, SPACE_SIZE),
                dtype=np.int8
            ),
            "map_features_energy": spaces.Box(
                low=-1,
                high=Global.MAX_ENERGY_PER_TILE,
```

```
            shape=(SPACE_SIZE, SPACE_SIZE),
            dtype=np.int8
        ),
        "relic_nodes_mask": spaces.Box(
            low=0,
            high=1,
            shape=(MAX_RELIC_NODES,),
            dtype=np.int8
        ),
        "relic_nodes": spaces.Box(
            low=-1,
            high=SPACE_SIZE - 1,
            shape=(MAX_RELIC_NODES, 2),
            dtype=np.int32
        ),
        "team_points": spaces.Box(
            low=0,
            high=1000,
            shape=(NUM_TEAMS,),
            dtype=np.int32
        ),
        "team_wins": spaces.Box(
            low=0,
            high=1000,
            shape=(NUM_TEAMS,),
            dtype=np.int32
        ),
        "steps": spaces.Box(
            low=0, high=Global.MAX_STEPS_IN_MATCH, shape=(1,),
dtype=np.int32
        ),
        "match_steps": spaces.Box(
            low=0, high=Global.MAX_STEPS_IN_MATCH, shape=(1,),
dtype=np.int32
        ),
        "remainingOverageTime": spaces.Box(
            low=0, high=1000, shape=(1,), dtype=np.int32
        ),
        "env_cfg_map_width": spaces.Box(
            low=0, high=SPACE_SIZE, shape=(1,), dtype=np.int32
        ),
        "env_cfg_map_height": spaces.Box(
```

```
            low=0, high=SPACE_SIZE, shape=(1,), dtype=np.int32
        ),
        "env_cfg_max_steps_in_match": spaces.Box(
            low=0, high=Global.MAX_STEPS_IN_MATCH, shape=(1,),
dtype=np.int32
        ),
        "env_cfg_unit_move_cost": spaces.Box(
            low=0, high=100, shape=(1,), dtype=np.int32
        ),
        "env_cfg_unit_sap_cost": spaces.Box(
            low=0, high=100, shape=(1,), dtype=np.int32
        ),
        "env_cfg_unit_sap_range": spaces.Box(
            low=0, high=100, shape=(1,), dtype=np.int32
        )
    })

    self.max_steps = Global.MAX_STEPS_IN_MATCH
    self.current_step = 0

    # 全图状态：地图瓦片、遗迹标记、能量地图
    self.tile_map = None     # -1未知、0空地、1星云、2小行星
    self.relic_map = None    # relic 存在标记，1 表示存在
    self.energy_map = None   # 每个 tile 的能量值

    # 单位状态：己方和敌方单位列表，每个单位以字典表示 {"x": int, "y":
int, "energy": int}
    self.team_units = []    # 己方
    self.enemy_units = []   # 敌方

    # 出生点：己方出生于左上角，敌方出生于右下角
    self.team_spawn = (0, 0)
    self.enemy_spawn = (SPACE_SIZE - 1, SPACE_SIZE - 1)

    # 探索记录：全图布尔数组，记录己方联合视野中已见过的 tile（全局只
记录一次）
    self.visited = None

    # 团队得分（己方得分）
    self.score = 0

    # 模拟环境的部分参数（env_cfg）
```

```
        self.env_cfg = {
            "map_width": SPACE_SIZE,
            "map_height": SPACE_SIZE,
            "max_steps_in_match": Global.MAX_STEPS_IN_MATCH,
            "unit_move_cost": Global.UNIT_MOVE_COST,
            "unit_sap_cost": Global.UNIT_SAP_COST if hasattr(Global,
"UNIT_SAP_COST") else 30,
            "unit_sap_range": Global.UNIT_SAP_RANGE,
        }

        # 新增：用于 relic 配置相关奖励
        self.relic_configurations = []   # list of (center_x, center_y,
mask(5x5 bool))
        self.potential_visited = None      # 全图记录，shape (SPACE_SIZE,
SPACE_SIZE)
        self.team_points_space = None      # 全图记录，哪些格子已经贡献过
team point

        self._init_state()

    def _init_state(self):
        """初始化全图状态、单位和记录"""
        num_tiles = SPACE_SIZE * SPACE_SIZE

        # 初始化 tile_map：随机部分设为 Nebula (1) 或 Asteroid (2)
        self.tile_map = np.zeros((SPACE_SIZE, SPACE_SIZE),
dtype=np.int8)
        num_nebula = int(num_tiles * 0.1)
        num_asteroid = int(num_tiles * 0.1)
        indices = np.random.choice(num_tiles, num_nebula +
num_asteroid, replace=False)
        flat_tiles = self.tile_map.flatten()
        flat_tiles[indices[:num_nebula]] = 1
        flat_tiles[indices[num_nebula:]] = 2
        self.tile_map = flat_tiles.reshape((SPACE_SIZE, SPACE_SIZE))

        # 初始化 relic_map：随机选取 3 个位置设置为 1（表示存在 relic）
        self.relic_map = np.zeros((SPACE_SIZE, SPACE_SIZE),
dtype=np.int8)
        relic_indices = np.random.choice(num_tiles, 3, replace=False)
        flat_relic = self.relic_map.flatten()
```

```
        flat_relic[relic_indices] = 1
        self.relic_map = flat_relic.reshape((SPACE_SIZE, SPACE_SIZE))

        # 初始化 energy_map：随机生成 2 个能量节点，值设为 MAX_ENERGY_PER_TILE，其余为 0
        self.energy_map = np.zeros((SPACE_SIZE, SPACE_SIZE), dtype=np.int8)
        num_energy_nodes = 2
        indices_energy = np.random.choice(num_tiles, num_energy_nodes, replace=False)
        flat_energy = self.energy_map.flatten()
        for idx in indices_energy:
            flat_energy[idx] = Global.MAX_ENERGY_PER_TILE
        self.energy_map = flat_energy.reshape((SPACE_SIZE, SPACE_SIZE))

        # 初始化己方单位：初始生成 1 个单位，出生于 team_spawn
        self.team_units = []
        spawn_x, spawn_y = self.team_spawn
        self.team_units.append({"x": spawn_x, "y": spawn_y, "energy": 100})

        # 初始化敌方单位：初始生成 1 个单位，出生于 enemy_spawn
        self.enemy_units = []
        spawn_x_e, spawn_y_e = self.enemy_spawn
        self.enemy_units.append({"x": spawn_x_e, "y": spawn_y_e, "energy": 100})

        # 初始化探索记录：全图大小，取各己方单位联合视野后标记已见区域
        self.visited = np.zeros((SPACE_SIZE, SPACE_SIZE), dtype=bool)
        union_mask = self.get_global_sensor_mask()
        self.visited = union_mask.copy()

        # 初始化 team score
        self.score = 0

        # 新增：初始化 relic 配置，及潜力点记录
        self.relic_configurations = []
        relic_coords = np.argwhere(self.relic_map == 1)
        for (y, x) in relic_coords:
            # 生成一个 5x5 mask，随机选择 5 个格子为 True（训练的时候选10
```

```
个吧，避免奖励太过于稀疏）
            mask = np.zeros((5,5), dtype=bool)
            indices = np.random.choice(25, 8, replace=False)
            mask_flat = mask.flatten()
            mask_flat[indices] = True
            mask = mask_flat.reshape((5,5))
            self.relic_configurations.append((x, y, mask))
        self.potential_visited = np.zeros((SPACE_SIZE, SPACE_SIZE),
dtype=bool)
        self.team_points_space = np.zeros((SPACE_SIZE, SPACE_SIZE),
dtype=bool)

        self.current_step = 0

    def compute_unit_vision(self, unit):
        """
        根据传入 unit 的位置计算其独立的 sensor mask,
        计算范围为单位传感器范围（切比雪夫距离），并对 Nebula tile 减少贡
献。
        取消环绕，只有在地图内的 tile 才计算。
        返回布尔矩阵 shape (SPACE_SIZE, SPACE_SIZE)。
        """
        sensor_range = Global.UNIT_SENSOR_RANGE
        nebula_reduction = 2
        vision = np.zeros((SPACE_SIZE, SPACE_SIZE), dtype=np.float32)
        x, y = unit["x"], unit["y"]
        for dy in range(-sensor_range, sensor_range + 1):
            for dx in range(-sensor_range, sensor_range + 1):
                new_x = x + dx
                new_y = y + dy
                if not (0 <= new_x < SPACE_SIZE and 0 <= new_y <
SPACE_SIZE):
                    continue
                contrib = sensor_range + 1 - max(abs(dx), abs(dy))
                if self.tile_map[new_y, new_x] == 1:
                    contrib -= nebula_reduction
                vision[new_y, new_x] += contrib
        return vision > 0

    def get_global_sensor_mask(self):
        """
```

```
        返回己方所有单位 sensor mask 的联合（逻辑 OR）。
        """
        mask = np.zeros((SPACE_SIZE, SPACE_SIZE), dtype=bool)
        for unit in self.team_units:
            mask |= self.compute_unit_vision(unit)
        return mask

    def get_unit_obs(self, unit):
        """
        根据传入 unit 的独立 sensor mask 构造局部观察字典,
        格式与比赛返回固定 JSON 格式相同。
        仅使用该 unit 自己能看到的区域进行过滤。
        """
        sensor_mask = self.compute_unit_vision(unit)
        map_tile_type = np.where(sensor_mask, self.tile_map, -1)
        map_energy = np.where(sensor_mask, self.energy_map, -1)
        map_features = {"tile_type": map_tile_type, "energy":
map_energy}
        sensor_mask_int = sensor_mask.astype(np.int8)

        # 构造单位信息，分别对己方与敌方单位过滤（使用该 unit 的 sensor
mask）
        units_position = np.full((NUM_TEAMS, MAX_UNITS, 2), -1,
dtype=np.int32)
        units_energy = np.full((NUM_TEAMS, MAX_UNITS, 1), -1,
dtype=np.int32)
        units_mask = np.zeros((NUM_TEAMS, MAX_UNITS),
dtype=np.int8)
        for i, u in enumerate(self.team_units):
            ux, uy = u["x"], u["y"]
            if sensor_mask[uy, ux]:
                units_position[0, i] = np.array([ux, uy])
                units_energy[0, i] = u["energy"]
                units_mask[0, i] = 1
        for i, u in enumerate(self.enemy_units):
            ux, uy = u["x"], u["y"]
            if sensor_mask[uy, ux]:
                units_position[1, i] = np.array([ux, uy])
                units_energy[1, i] = u["energy"]
                units_mask[1, i] = 1
```

```
        units = {"position": units_position, "energy": units_energy}

        # 构造 relic_nodes 信息：仅显示在 sensor_mask 内的 relic 坐标
        relic_coords = np.argwhere(self.relic_map == 1)
        relic_nodes = np.full((MAX_RELIC_NODES, 2), -1,
dtype=np.int32)
        relic_nodes_mask = np.zeros(MAX_RELIC_NODES,
dtype=np.int8)
        idx = 0
        for (ry, rx) in relic_coords:
            if idx >= MAX_RELIC_NODES:
                break
            if sensor_mask[ry, rx]:
                relic_nodes[idx] = np.array([rx, ry])
                relic_nodes_mask[idx] = 1
            else:
                relic_nodes[idx] = np.array([-1, -1])
                relic_nodes_mask[idx] = 0
            idx += 1

        team_points = np.array([self.score, 0], dtype=np.int32)
        team_wins = np.array([0, 0], dtype=np.int32)
        steps = self.current_step
        match_steps = self.current_step

        obs = {
            "units": units,
            "units_mask": units_mask,
            "sensor_mask": sensor_mask_int,
            "map_features": map_features,
            "relic_nodes_mask": relic_nodes_mask,
            "relic_nodes": relic_nodes,
            "team_points": team_points,
            "team_wins": team_wins,
            "steps": steps,
            "match_steps": match_steps
        }
        observation = {
            "obs": obs,
            "remainingOverageTime": 60,
```

```
            "player": "player_0",
            "info": {"env_cfg": self.env_cfg}
        }
        return observation

    def get_obs(self):
        """
        返回平铺后的全局观测字典，确保所有键与 observation_space 完全一致。
        """
        sensor_mask = self.get_global_sensor_mask()
        sensor_mask_int = sensor_mask.astype(np.int8)

        map_features_tile_type = np.where(sensor_mask, self.tile_map, -1)
        map_features_energy = np.where(sensor_mask, self.energy_map, -1)

        units_position = np.full((NUM_TEAMS, MAX_UNITS, 2), -1, dtype=np.int32)
        units_energy = np.full((NUM_TEAMS, MAX_UNITS, 1), -1, dtype=np.int32)
        units_mask = np.zeros((NUM_TEAMS, MAX_UNITS), dtype=np.int8)

        # 己方单位
        for i, unit in enumerate(self.team_units):
            ux, uy = unit["x"], unit["y"]
            if sensor_mask[uy, ux]:
                units_position[0, i] = np.array([ux, uy])
                units_energy[0, i] = unit["energy"]
                units_mask[0, i] = 1
        # 敌方单位
        for i, unit in enumerate(self.enemy_units):
            ux, uy = unit["x"], unit["y"]
            if sensor_mask[uy, ux]:
                units_position[1, i] = np.array([ux, uy])
                units_energy[1, i] = unit["energy"]
                units_mask[1, i] = 1
```

```
        relic_coords = np.argwhere(self.relic_map == 1)
        relic_nodes = np.full((MAX_RELIC_NODES, 2), -1,
dtype=np.int32)
        relic_nodes_mask = np.zeros((MAX_RELIC_NODES,),
dtype=np.int8)
        idx = 0
        for (ry, rx) in relic_coords:
            if idx >= MAX_RELIC_NODES:
                break
            if sensor_mask[ry, rx]:
                relic_nodes[idx] = np.array([rx, ry])
                relic_nodes_mask[idx] = 1
            else:
                relic_nodes[idx] = np.array([-1, -1])
                relic_nodes_mask[idx] = 0
            idx += 1

        team_points = np.array([self.score, 0], dtype=np.int32)
        team_wins = np.array([0, 0], dtype=np.int32)
        steps = np.array([self.current_step], dtype=np.int32)
        match_steps = np.array([self.current_step], dtype=np.int32)
        remainingOverageTime = np.array([60], dtype=np.int32)

        env_cfg_map_width = np.array([self.env_cfg["map_width"]],
dtype=np.int32)
        env_cfg_map_height = np.array([self.env_cfg["map_height"]],
dtype=np.int32)
        env_cfg_max_steps_in_match =
np.array([self.env_cfg["max_steps_in_match"]], dtype=np.int32)
        env_cfg_unit_move_cost =
np.array([self.env_cfg["unit_move_cost"]], dtype=np.int32)
        env_cfg_unit_sap_cost =
np.array([self.env_cfg["unit_sap_cost"]], dtype=np.int32)
        env_cfg_unit_sap_range =
np.array([self.env_cfg["unit_sap_range"]], dtype=np.int32)

        flat_obs = {
            "units_position": units_position,
```

```
            "units_energy": units_energy,
            "units_mask": units_mask,
            "sensor_mask": sensor_mask_int,
            "map_features_tile_type": map_features_tile_type,
            "map_features_energy": map_features_energy,
            "relic_nodes_mask": relic_nodes_mask,
            "relic_nodes": relic_nodes,
            "team_points": team_points,
            "team_wins": team_wins,
            "steps": steps,
            "match_steps": match_steps,
            "remainingOverageTime": remainingOverageTime,
            "env_cfg_map_width": env_cfg_map_width,
            "env_cfg_map_height": env_cfg_map_height,
            "env_cfg_max_steps_in_match":
env_cfg_max_steps_in_match,
            "env_cfg_unit_move_cost": env_cfg_unit_move_cost,
            "env_cfg_unit_sap_cost": env_cfg_unit_sap_cost,
            "env_cfg_unit_sap_range": env_cfg_unit_sap_range
        }

        return flat_obs

    def reset(self):
        """
        重置环境状态，并返回初始的平铺观测数据。
        """
        self._init_state()
        return self.get_obs()

    def _spawn_unit(self, team):
        """生成新单位：己方或敌方，初始能量 100，出生于各自出生点"""
        if team == 0:
            spawn_x, spawn_y = self.team_spawn
            self.team_units.append({"x": spawn_x, "y": spawn_y,
"energy": 100})
        elif team == 1:
            spawn_x, spawn_y = self.enemy_spawn
            self.enemy_units.append({"x": spawn_x, "y": spawn_y,
"energy": 100})

    def step(self, actions):
```

```
        """
        根据动作更新环境状态，并返回 (observation, reward, done, info)。
        修改后的奖励逻辑:
        1. 每个 unit 单独计算 unit_reward。
        2. 若移动动作导致超出地图或目标 tile 为 Asteroid，则判定为无效，unit_reward -0.2。
        3. Sap 动作:
            - 检查 unit 局部 obs 中 relic_nodes_mask 是否存在 relic；
            - 如果存在，统计 unit 8 邻域内敌方单位数，若数目>=2，则 sap 奖励 = +1.0× 敌方单位数，否则扣 -2.0；
            - 若无 relic 可见，则同样扣 -2.0。
        4. 非 sap 动作:
            - 成功移动后，检查该 unit 是否位于任一 relic 配置内的潜力点：
                * 若首次访问该潜力点，unit_reward +2.0，并标记 visited；
                * 如果该潜力点尚未兑现 team point，则增加 self.score 1，同时 unit_reward +3.0 并标记为 team_points_space；
                * 如果已在 team_points_space 上，则每回合奖励 +3.0；
            - 若 unit 位于能量节点（energy == Global.MAX_ENERGY_PER_TILE），unit_reward +0.2；
            - 若 unit 位于 Nebula（tile_type==1），unit_reward -0.2；
            - 如果 unit 移动后与敌方 unit 重合，且对方能量低于己方，则对每个满足条件的敌方 unit 奖励 +1.0。
        5. 全局探索奖励：所有己方单位联合视野中新发现 tile，每个奖励 +0.1。
        6. 每一step结束，奖励 point*0.3的奖励 + 规则*0.7的奖励
        7. 每 3 步生成新单位；每 20 步整体滚动地图和敌方单位位置（滚动时对敌方单位使用边界检查）。
        """
        prev_score = self.score

        self.current_step += 1
        total_reward = 0.0

        # 处理每个己方单位
        for idx, unit in enumerate(self.team_units):
            unit_reward = 0.0
            act = actions[idx]
            action_enum = ActionType(act)
            # print(f"Unit {idx} action: {action_enum}",file=sys.stderr)

            # 获取该 unit 的局部 obs
            unit_obs = self.get_unit_obs(unit)
```

```
            # 如果动作为 sap
            if action_enum == ActionType.sap:
                # 检查局部 obs 中是否有 relic 可见
                if np.any(unit_obs["obs"]["relic_nodes_mask"] == 1):
                    # 统计 unit 周围 8 邻域内敌方单位数
                    enemy_count = 0
                    for dy in [-1, 0, 1]:
                        for dx in [-1, 0, 1]:
                            if dx == 0 and dy == 0:
                                continue
                            nx = unit["x"] + dx
                            ny = unit["y"] + dy
                            if not (0 <= nx < SPACE_SIZE and 0 <= ny <
SPACE_SIZE):
                                continue
                            for enemy in self.enemy_units:
                                if enemy["x"] == nx and enemy["y"] == ny:
                                    enemy_count += 1
                    if enemy_count >= 2:
                        unit_reward += 1.0 * enemy_count
                    else:
                        unit_reward -= 1.0
                else:
                    unit_reward -= 1.0
                # Sap 动作不改变位置
            else:
                # 计算移动方向
                if action_enum in [ActionType.up, ActionType.right,
ActionType.down, ActionType.left]:
                    dx, dy = action_enum.to_direction()
                else:
                    dx, dy = (0, 0)
                new_x = unit["x"] + dx
                new_y = unit["y"] + dy
                # 检查边界和障碍
                if not (0 <= new_x < SPACE_SIZE and 0 <= new_y <
SPACE_SIZE):
                    new_x, new_y = unit["x"], unit["y"]
```

```
                unit_reward -= 0.2  # 超出边界
            elif self.tile_map[new_y, new_x] == 2:
                new_x, new_y = unit["x"], unit["y"]
                unit_reward -= 0.2  # 遇到 Asteroid
            else:
                # 移动成功
                unit["x"], unit["y"] = new_x, new_y

            # 重新获取移动后的局部 obs
            unit_obs = self.get_unit_obs(unit)

            # 检查 relic 配置奖励：遍历所有 relic 配置，判断该 unit 是否位于配置中（计算时考虑边界）
            for (rx, ry, mask) in self.relic_configurations:
                # relic 配置区域范围：中心 (rx, ry) ±2
                # 如果 unit 在 [rx-2, rx+2] 和 [ry-2, ry+2] 范围内
                if rx - 2 <= unit["x"] <= rx + 2 and ry - 2 <= unit["y"] <= ry + 2:
                    # 计算在配置 mask 中的索引
                    ix = unit["x"] - rx + 2
                    iy = unit["y"] - ry + 2
                    # 检查索引是否在 mask 范围内（考虑边界）
                    if 0 <= ix < 5 and 0 <= iy < 5:
                        # 如果该潜力点未被访问，则奖励 +2.0
                        if not mask[iy, ix]:
                            if not self.potential_visited[unit["y"], unit["x"]]:
                                unit_reward += 1.5
                                self.potential_visited[unit["y"], unit["x"]] = True
                        # 如果潜力点是真的points_space，则奖励 +3.0
                        else:
                            # 如果该点尚未产生 team point，则增加 team point并奖励 +3.0
                            if not self.team_points_space[unit["y"], unit["x"]]:
                                self.score += 1
                                unit_reward += 3.0
                                self.team_points_space[unit["y"], unit["x"]] = True
                            else:
                                # 已在 team_points_space 上，每回合奖励 +3.0
```

```
                        self.score += 1
                        unit_reward += 3.0
            # 能量节点奖励
            if unit_obs["obs"]["map_features"]["energy"][unit["y"],
unit["x"]] == Global.MAX_ENERGY_PER_TILE:
                unit_reward += 0.2
            # Nebula 惩罚
            if unit_obs["obs"]["map_features"]["tile_type"][unit["y"],
unit["x"]] == 1:
                unit_reward -= 0.2
            # 攻击行为：若与敌方单位重合且对方能量低于己方，则对每个敌人
奖励 +1.0
            for enemy in self.enemy_units:
                if enemy["x"] == unit["x"] and enemy["y"] == unit["y"]:
                    if enemy["energy"] < unit["energy"]:
                        unit_reward += 1.0
            total_reward += unit_reward
            #
print("##################################",file=sys.stderr)
            # print("step:",self.current_step)
            # print("")
            # print(total_reward,file=sys.stderr)

        # 全局探索奖励：利用所有己方单位联合视野中新发现的 tile
        union_mask = self.get_global_sensor_mask()
        new_tiles = union_mask & (~self.visited)
        num_new = np.sum(new_tiles)
        if num_new > 0:
            total_reward += 0.2 * num_new
        self.visited[new_tiles] = True


        # 每 3 步生成新单位（若未达到 MAX_UNITS）
        if self.current_step % 3 == 0:
            if len(self.team_units) < MAX_UNITS:
                self._spawn_unit(team=0)
            if len(self.enemy_units) < MAX_UNITS:
                self._spawn_unit(team=1)

        # 每 20 步整体滚动地图、遗迹和能量图，以及敌方单位位置（右移 1
格，边界检查）
```

```
        if self.current_step % 20 == 0:
            # 这里采用 np.roll 保持地图内部数据不变，但对于敌方单位，我们检查边界
            self.tile_map = np.roll(self.tile_map, shift=1, axis=1)
            self.relic_map = np.roll(self.relic_map, shift=1, axis=1)
            self.energy_map = np.roll(self.energy_map, shift=1, axis=1)
            for enemy in self.enemy_units:
                new_ex = enemy["x"] + 1
                if new_ex >= SPACE_SIZE:
                    new_ex = enemy["x"]  # 保持不变
                enemy["x"] = new_ex

        # 在 step 结束时计算 self.score 的增加量
        score_increase = self.score - prev_score

        # 将总奖励合并：total_reward * 0.5 + score_increase * 0.5
        final_reward = total_reward * 0.5 + score_increase * 0.5
        # final_reward =score_increase

        done = self.current_step >= self.max_steps
        # done = self.current_step >= 200
        info = {"score": self.score, "step": self.current_step}
        return self.get_obs(), final_reward, done, info

    def render(self, mode='human'):
        display = self.tile_map.astype(str).copy()
        for unit in self.team_units:
            display[unit["y"], unit["x"]] = 'A'
        print("Step:", self.current_step)
        print(display)
```

train.py

In [ ]:

```
%%writefile agent/train.py

from stable_baselines3 import PPO
from ppo_game_env import PPOGameEnv

# 创建环境实例
env = PPOGameEnv()
```

```
# 使用多层感知机策略初始化 PPO 模型
# model = PPO("MultiInputPolicy",
env,learning_rate=0.0005,ent_coef=0.1,vf_coef = 0.3, verbose=1)
model = PPO("MultiInputPolicy", env,learning_rate=0.0005,
verbose=1)
# model_1 = PPO("MultiInputPolicy", env,learning_rate=0.0005,
verbose=1)



# total_timesteps may need to adjust
# model.learn(total_timesteps=960000)
model.learn(total_timesteps=6000)

# 保存训练好的模型
model.save("/kaggle/working/agent/ppo_game_env_model")

# 测试：加载模型并进行一次模拟
# obs = env.reset()
# done = False
# while not done:
#     action, _ = model.predict(obs)
#     obs, reward, done, info = env.step(action)
#     env.render()
```

agent.py

In [ ]:

```
import os
import sys
import numpy as np
from stable_baselines3 import PPO

def transform_obs(comp_obs, env_cfg=None):
    """
    将比赛引擎返回的 JSON 观测转换为模型训练时使用的平铺观测格式。

    比赛环境的观测格式（comp_obs）结构如下：
     {
       "obs": {
          "units": {"position": Array(T, N, 2), "energy": Array(T, N, 1)},
          "units_mask": Array(T, N),
```

```
        "sensor_mask": Array(W, H),
        "map_features": {"energy": Array(W, H), "tile_type": Array(W,
H)},
        "relic_nodes_mask": Array(R),
        "relic_nodes": Array(R, 2),
        "team_points": Array(T),
        "team_wins": Array(T),
        "steps": int,
        "match_steps": int
      },
      "remainingOverageTime": int,
      "player": str,
      "info": {"env_cfg": dict}
    }
```

我们需要构造如下平铺字典（与 *PPOGameEnv.get_obs()* 返回的格式一致）：

```
    {
      "units_position": (T, N, 2),
      "units_energy": (T, N, 1),
      "units_mask": (T, N),
      "sensor_mask": (W, H),
      "map_features_tile_type": (W, H),
      "map_features_energy": (W, H),
      "relic_nodes_mask": (R,),
      "relic_nodes": (R, 2),
      "team_points": (T,),
      "team_wins": (T,),
      "steps": (1,),
      "match_steps": (1,),
      "remainingOverageTime": (1,),
      "env_cfg_map_width": (1,),
      "env_cfg_map_height": (1,),
      "env_cfg_max_steps_in_match": (1,),
      "env_cfg_unit_move_cost": (1,),
      "env_cfg_unit_sap_cost": (1,),
      "env_cfg_unit_sap_range": (1,)
    }
    """
    # 如果存在 "obs" 键，则取其内部数据，否则直接使用 comp_obs
    if "obs" in comp_obs:
        base_obs = comp_obs["obs"]
```

```
    else:
        base_obs = comp_obs


    flat_obs = {}

    # 处理 units 数据
    if "units" in base_obs:
        flat_obs["units_position"] = np.array(base_obs["units"]
["position"], dtype=np.int32)
        flat_obs["units_energy"] = np.array(base_obs["units"]["energy"],
dtype=np.int32)
        # 如果 units_energy 的 shape 为 (NUM_TEAMS, MAX_UNITS) 则扩
展一个维度
        if flat_obs["units_energy"].ndim == 2:
            flat_obs["units_energy"] =
np.expand_dims(flat_obs["units_energy"], axis=-1)
    else:
        flat_obs["units_position"] = np.array(base_obs["units_position"],
dtype=np.int32)
        flat_obs["units_energy"] = np.array(base_obs["units_energy"],
dtype=np.int32)
        if flat_obs["units_energy"].ndim == 2:
            flat_obs["units_energy"] =
np.expand_dims(flat_obs["units_energy"], axis=-1)


    # 处理 units_mask
    if "units_mask" in base_obs:
        flat_obs["units_mask"] = np.array(base_obs["units_mask"],
dtype=np.int8)
    else:
        flat_obs["units_mask"] =
np.zeros(flat_obs["units_position"].shape[:2], dtype=np.int8)


    # 处理 sensor_mask：若返回的是 3D 数组，则取逻辑 or 得到全局 mask
    sensor_mask_arr = np.array(base_obs["sensor_mask"],
dtype=np.int8)
    if sensor_mask_arr.ndim == 3:
```

```
            sensor_mask = np.any(sensor_mask_arr, axis=0).astype(np.int8)
        else:
            sensor_mask = sensor_mask_arr
        flat_obs["sensor_mask"] = sensor_mask

        # 处理 map_features（tile_type 与 energy）
        if "map_features" in base_obs:
            mf = base_obs["map_features"]
            flat_obs["map_features_tile_type"] = np.array(mf["tile_type"],
dtype=np.int8)
            flat_obs["map_features_energy"] = np.array(mf["energy"],
dtype=np.int8)
        else:
            flat_obs["map_features_tile_type"] =
np.array(base_obs["map_features_tile_type"], dtype=np.int8)
            flat_obs["map_features_energy"] =
np.array(base_obs["map_features_energy"], dtype=np.int8)

        # 处理 relic 节点信息
        if "relic_nodes_mask" in base_obs:
            flat_obs["relic_nodes_mask"] =
np.array(base_obs["relic_nodes_mask"], dtype=np.int8)
        else:
            max_relic = env_cfg.get("max_relic_nodes", 6) if env_cfg is not
None else 6
            flat_obs["relic_nodes_mask"] = np.zeros((max_relic,),
dtype=np.int8)
        if "relic_nodes" in base_obs:
            flat_obs["relic_nodes"] = np.array(base_obs["relic_nodes"],
dtype=np.int32)
        else:
            max_relic = env_cfg.get("max_relic_nodes", 6) if env_cfg is not
None else 6
            flat_obs["relic_nodes"] = np.full((max_relic, 2), -1,
dtype=np.int32)

        # 处理团队得分与胜局
        if "team_points" in base_obs:
            flat_obs["team_points"] = np.array(base_obs["team_points"],
```

```
dtype=np.int32)
    else:
        flat_obs["team_points"] = np.zeros(2, dtype=np.int32)
    if "team_wins" in base_obs:
        flat_obs["team_wins"] = np.array(base_obs["team_wins"],
dtype=np.int32)
    else:
        flat_obs["team_wins"] = np.zeros(2, dtype=np.int32)

    # 处理步数信息
    if "steps" in base_obs:
        flat_obs["steps"] = np.array([base_obs["steps"]],
dtype=np.int32)
    else:
        flat_obs["steps"] = np.array([0], dtype=np.int32)
    if "match_steps" in base_obs:
        flat_obs["match_steps"] = np.array([base_obs["match_steps"]],
dtype=np.int32)
    else:
        flat_obs["match_steps"] = np.array([0], dtype=np.int32)

    # 注意：不在此处处理 remainingOverageTime,
    # 将在 Agent.act 中利用传入的参数添加

    # 补全环境配置信息
    if env_cfg is not None:
        flat_obs["env_cfg_map_width"] =
np.array([env_cfg["map_width"]], dtype=np.int32)
        flat_obs["env_cfg_map_height"] =
np.array([env_cfg["map_height"]], dtype=np.int32)
        flat_obs["env_cfg_max_steps_in_match"] =
np.array([env_cfg["max_steps_in_match"]], dtype=np.int32)
        flat_obs["env_cfg_unit_move_cost"] =
np.array([env_cfg["unit_move_cost"]], dtype=np.int32)
        flat_obs["env_cfg_unit_sap_cost"] =
np.array([env_cfg["unit_sap_cost"]], dtype=np.int32)
        flat_obs["env_cfg_unit_sap_range"] =
np.array([env_cfg["unit_sap_range"]], dtype=np.int32)
```

```
    else:
        flat_obs["env_cfg_map_width"] = np.array([0], dtype=np.int32)
        flat_obs["env_cfg_map_height"] = np.array([0], dtype=np.int32)
        flat_obs["env_cfg_max_steps_in_match"] = np.array([0],
dtype=np.int32)
        flat_obs["env_cfg_unit_move_cost"] = np.array([0],
dtype=np.int32)
        flat_obs["env_cfg_unit_sap_cost"] = np.array([0],
dtype=np.int32)
        flat_obs["env_cfg_unit_sap_range"] = np.array([0],
dtype=np.int32)

    return flat_obs

class Agent():
    def __init__(self, player: str, env_cfg) -> None:
        self.player = player
        self.opp_player = "player_1" if self.player == "player_0" else
"player_0"
        self.team_id = 0 if self.player == "player_0" else 1
        self.opp_team_id = 1 if self.team_id == 0 else 0
        np.random.seed(0)
        self.env_cfg = env_cfg

        # 如果 env_cfg 中没有 "max_units"，则补上默认值 16
        if "max_units" not in self.env_cfg:
            self.env_cfg["max_units"] = 16


        # 加载训练好的 PPO 模型（请确保模型文件路径正确）
        model_path = os.path.join(os.path.dirname(__file__),
"ppo_game_env_model.zip")
        self.model = PPO.load(model_path)

    def act(self, step: int, obs, remainingOverageTime: int = 60):
        """
        根据比赛观测与当前步数决定各单位动作。
        输出为形状 (max_units, 3) 的 numpy 数组，每行格式为 [动作类型,
delta_x, delta_y]，
        其中非汲取动作时 delta_x 和 delta_y 固定为 0。
        """
```

```
        import sys
        # # 当 step 为 11 时打印调试信息
        # if step == 11:
        #     print("DEBUG: Agent.act() 调用参数：", file=sys.stderr)
        #     print("DEBUG: self.player =", self.player, file=sys.stderr)
        #     print("DEBUG: step =", step, file=sys.stderr)
        #     # 打印 obs 的 key 列表，可以查看观测数据的大致结构
        #     print("DEBUG: obs keys =", list(obs.keys()), file=sys.stderr)
        #
print("================================================================", file=sys.stderr)
        #     print("DEBUG: ob =", obs, file=sys.stderr)
        #     print("DEBUG: remainingOverageTime =",
remainingOverageTime, file=sys.stderr)
        #
print("################################################################", file=sys.stderr)

        flat_obs = transform_obs(obs, self.env_cfg)
        # 如果当前 agent 为 player_1，则交换单位信息（确保和训练时候一
致，己方视角永远在第一位置）
        if self.player == "player_1":
            flat_obs["units_position"] = flat_obs["units_position"][::-1]
            flat_obs["units_energy"] = flat_obs["units_energy"][::-1]
            flat_obs["units_mask"] = flat_obs["units_mask"][::-1]

        # 手动添加 remainingOverageTime（取自传入参数）
        flat_obs["remainingOverageTime"] =
np.array([remainingOverageTime], dtype=np.int32)

        # if step == 11:
        #
print("----------------------------------------------------------",
file=sys.stderr)
        #     print("DEBUG: flat_obs =", flat_obs, file=sys.stderr)
        #
print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^",
file=sys.stderr)
        # 使用模型预测动作（deterministic 模式）
        action, _ = self.model.predict(flat_obs, deterministic=True)
        # 确保 action 为 numpy 数组，并显式设置为 np.int32 类型
```

```
        action = np.array(action, dtype=np.int32)

        max_units = self.env_cfg["max_units"]
        actions = np.zeros((max_units, 3), dtype=np.int32)
        for i, a in enumerate(action):
            actions[i, 0] = int(a)
            actions[i, 1] = 0  # 若为 sap 动作，可在此扩展目标偏移
            actions[i, 2] = 0
        return actions
```

main.py

In [ ]:

```
%%writefile agent/main.py

import json
from typing import Dict
import sys
from argparse import Namespace

import numpy as np

from agent import Agent
# from lux.config import EnvConfig
from lux.kit import from_json
### DO NOT REMOVE THE FOLLOWING CODE ###
agent_dict = dict() # store potentially multiple dictionaries as kaggle imports code directly
agent_prev_obs = dict()
def agent_fn(observation, configurations):
    """
    agent definition for kaggle submission.
    """
    global agent_dict
    obs = observation.obs
    if type(obs) == str:
        obs = json.loads(obs)
    step = observation.step
    player = observation.player
    remainingOverageTime = observation.remainingOverageTime
    if step == 0:
        agent_dict[player] = Agent(player, configurations["env_cfg"])
```

```
    agent = agent_dict[player]
    actions = agent.act(step, from_json(obs), remainingOverageTime)
    return dict(action=actions.tolist())
if __name__ == "__main__":

    def read_input():
        """
        Reads input from stdin
        """
        try:
            return input()
        except EOFError as eof:
            raise SystemExit(eof)
    step = 0
    player_id = 0
    env_cfg = None
    i = 0
    while True:
        inputs = read_input()
        raw_input = json.loads(inputs)
        observation = Namespace(**dict(step=raw_input["step"],
obs=raw_input["obs"],
remainingOverageTime=raw_input["remainingOverageTime"],
player=raw_input["player"], info=raw_input["info"]))
        if i == 0:
            env_cfg = raw_input["info"]["env_cfg"]
            player_id = raw_input["player"]
        i += 1
        actions = agent_fn(observation, dict(env_cfg=env_cfg))
        # send actions to engine
        print(json.dumps(actions))
```

Test run

In [ ]:

```
!pip install --upgrade luxai-s3
```

In [ ]:

```
!python agent/train.py
```

In [ ]:

```
!luxai-s3 agent/main.py agent/main.py --seed 37 --
```

```
output=replay.html
```

In [ ]:

```
import IPython # load the HTML replay
IPython.display.HTML(filename='replay.html')
```

Create a submission

In [ ]:

```
!cd agent && tar -czf submission.tar.gz *
!mv agent/submission.tar.gz .
```

In [ ]: