Arenarium Documentation

Arenarium

Sep 03, 2019

Contents:

CHAPTER 1

Getting Started

You are probably here because you want to get started playing Arenarium. Arenarium is a game you play by writing code, and this guide explains how.

1.1 Rules

The rules of the basic Arena game are simple. The game starts with two or more gladiators (agents) inside a dungeon. On each turn, your gladiator has the option between three types of moves:

1. **Move** to a neighboring grid points.
2. **Attack** another gladiator within range.
3. **Stay** in the same place (and do nothing).

A fourth move will be coming soon :

(4.) **Boost** a stat of your gladiator to improve its accuracy, evasion, damage, protection or speed.

Objective: The last gladiator to survive wins.

Score: For each gladiator that your gladiator kills it receives one point. However, the score is set to zero if your gladiator dies.

1.2 Under the Hood

On a more detailed level, the Arena game is operating on an event queue system. Each move is queued and then processed after some time, which is influenced by a gladiator's speed. The base speed is 23 ticks. This is how much 'time' it takes for a queued move to resolve. After a move is resolved, the agent can queue the next move and so on.

Attacks are handled on a competitive d10 dice roll. The attacker and defender each roll a die, to which attacker adds their accuracy (base 0) and the defender their evasion (base 0). If the attacker's total is at least as high as the defender's one, the attack hits. The amount of lost hit points of the defender is given by subtracting the defender's protection (base 0) from the attacker's damage (base 5), to a minimum of zero.

(Coming soon:) Boosts let you change certain stats of your gladiator. Improve your accuracy or evasion, hit harder by adding some points to your damage, reduce the amount of damage you take by strengthening your protection, or get faster by improving your speed. For that, each gladiator has ten spirit points, which can be allocated to boost the aforementioned stats. Raising a stat by one point costs one spirit point, raising it by two however costs three, raising it by three costs six, and raising it by four points costs ten spirit points. You can also lower previously raised stats again to free up spirit points and re-allocate them. Though all of this takes time and you will have to decide what is worth investing in!

1.3 Writing your own agent

In principle, all you need is a text editor. However, we recommend getting started with the agent development template because it allows you to test your agents locally before uploading it to the Arenarium website.

Once you have set up your environment, it is time to learn about the basic anatomy of an agent.

Every agent you write should derive from the *Agent* class. Don't worry, the only thing you have to implement is the *move* function. A minimal agent implementation would look like this:

```
from battleground.agent import Agent


class ArenaAgent(Agent):
    def move(self, state):
        """state is a dictionary representing the current game state."""

        # ... do something to read state ...

        move = {'type': 'stay', 'value': 1}
        return move
```

This agent just sits still for one turn. You are free to read the game state directly and process it however you like. However, it is easiest to start with the basic building blocks provided by the *building_blocks* module.

The following example aggressively attacks the nearest other player.

```
from battleground.agent import Agent
from battleground.games.arena import building_blocks


class ArenaAgent(Agent):

    def move(self, state):
        """
        Attack nearest other or move towards nearest other.
        """

        # try attack move is valid
        move = building_blocks.attack_closest(state)
        if move is not None:
            return move

        # if attack is not possible, move towards closest other
        closest = building_blocks.closest_other_location(state)
        move = building_blocks.move_toward(state, closest)
        if move is not None:
            return move
```

(continues on next page)

(continued from previous page)

```
        # if move is not possible, do nothing.
        return {}
```

From here it is up to you. Enjoy!

CHAPTER 2

Agent Memory

Agents have the ability to remember information from previous games. This enables them to learn and improve over time.

You can get/set this memory using the *get_memory()* and *set_memory()* methods of the agent class.

A simple example would look like this:

```
from battleground.agent import Agent


class PersistentAgent(Agent):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.default_mem = {"guess": 5}

    def move(self, state):
        memory = self.get_memory(default=self.default_mem)
        # do something with memory here
        my_move = {"value": memory["guess"]}

        # update memory
        memory['key 1'] = 'value _1'
        memory['key 2'] = 'value _2'
        self.set_memory(memory)

        # return move
        return my_move
```

CHAPTER 3

Modules

3.1 Battleground Agent

class battleground.agent.**Agent**(***kwargs*)

This is an interface for the agent class. Every agent should sub-class this class. The entrypoint for this class is the move() function.

get_memory(*default=None*)

This function can be used by an agent to get its persistent memory. This function is also used by site runner to get the agent's memory at the end of a game and store it in the database.

Parameters **default** – if the agent memory is not set, return the default value (None).

Returns the persistent memory of the agent.

move(*state*)

Main entry point for the agent class, agent logic goes here. This function is called by the game runner when it's this agent's turn to make a move.

Parameters **state** – the current game state.

Returns a valid move.

observe(*state*)

This function is called by the game engine every time an update to the game state is available. (Even on other player's turns.)

Parameters **state** – the current game state.

set_memory(*data*)
Set the persistent memory of the agent. This function should be called by the agent if the persistent memory needs to be updated. This function is also called by the site runner at the start of a game.

Parameters **data** – the data to save.

3.2 Game Engine

class battleground.game_engine.**GameEngine**(*num_players*, *type*, ***kwargs*)
This is an interface for the game engine class. An engine for a specific game should implement these functions.

game_over()
Check if the game is over.

Returns (bool) is the game over.

get_current_player()
This is used by the game runner to determine which player should make the next move

Returns (int) index of the current player in the players list of the GameRunner

get_game_name()

Returns (str) the type of the current game.

get_save_state()

Returns the state of the game as it should be saved in the database.

get_state()
Get the current state of the game.

Returns the current game state.

move(*move*)
Resolve a move in the game engine on behalf of the current player. This function is called by the game runner, taking the agent's chosen move of the current player.

Parameters **move** – the move returned by the agent of the current player.

reset()
Initialize the game to the starting point.

3.3 Arena Building Blocks

battleground.games.arena.building_blocks.**attack**(*state*, *target*)
Generate a move object to attack the target, if that is a valid move. Otherwise, return None.

Parameters

- **state** – The current game state.
- **target** – id of the target.

Returns A move object (dict) or None

battleground.games.arena.building_blocks.**attack_closest**(*state*)
Generate a move object to attack the closest other player, if that is a valid move. Otherwise, return None.

Parameters **state** – The current game state.

Returns A move object (dict), or None.

battleground.games.arena.building_blocks.**attack_myself**(*state*)
Generate a move object to attack yourself.

Parameters **state** – The current game state.

Returns A move object (dict).

battleground.games.arena.building_blocks.**closest_other**(*state*)
Get the id of the closest other player.

Parameters **state** – The current game state.

Returns (int) index in gladiator list.

battleground.games.arena.building_blocks.**closest_other_location**(*state*)
Get the location of the closest other player, e.g., (x, y).

Parameters **state** – The current game state.

Returns tuple (x, y).

battleground.games.arena.building_blocks.**distances**(*reference_location*, *locations*)
Compute distances from a reference location to a set of other locations.

Parameters

- **reference_location** – iterable of coordinates, e.g., (1, 2)
- **locations** – iterable of iterables of coordinates, e.g., [(0, 0), (2, 1), . . .]

Returns dict of distances {id: float, . . . }

battleground.games.arena.building_blocks.**move_away**(*state*, *location*)
Generate move that takes you most quickly away from the specified location.

Parameters

- **state** – The current game state.
- **location** – tuple of target position, e.g, (x, y)

Returns A move object (dict).

battleground.games.arena.building_blocks.**move_relative**(*state*, *location*, *towards*)
Generate move that takes you most directly towards or away from the specified location.

shorthand functions move_toward and move_away are available.

Parameters

- **state** – The current game state.
- **location** – tuple of target position, e.g, (x, y)
- **towards** – bool, move towards location if true, away otherwise

Returns A move object (dict).

battleground.games.arena.building_blocks.**move_toward**(*state*, *location*)
Generate move that takes you most directly to the specified location.

Parameters

- **state** – The current game state.
- **location** – tuple of target position, e.g, (x, y)

Returns A move object (dict).

battleground.games.arena.building_blocks.**my_hitpoints**(*state*)
Return current health (hitpoints) of the player.

Parameters **state** – The current game state.

Returns int.

battleground.games.arena.building_blocks.**my_location**(*state*)
Get the location of the current player.

Parameters **state** – The current game state.

Returns tuple (x, y)

battleground.games.arena.building_blocks.**others**(*state*, *alive=True*)
Get a dictionary of other players.

Parameters **state** – The current game state.

Returns (dict) {id: player data, . . . }

battleground.games.arena.building_blocks.**others_hitpoints**(*state*)
Return current health (hitpoints) of other players.

Parameters **state** – The current game state.

Returns dict(id: int).

battleground.games.arena.building_blocks.**others_locations**(*state*)
Get a dictionary of the locations of players, excluding the current player.

Parameters **state** – The current game state.

Returns dict of locations, e.g., {id: (x, y), . . . }.

CHAPTER 4

Indices and tables

- genindex
- modindex

Python Module Index

b

Index