

# Softserve Integration Guide

This guide explains how to integrate your project with the Softserve API. This will allow your project to participate in automated AI versus AI tournaments. It is required for the official midterm and final tournaments.

## Background

The Softserve web API allows projects to play each other in an automated and efficient way. Before Softserve, the class tournaments were conducted with manual move input and only involved a small number of games. With Softserve, games are played without any human intervention, and tournaments involve hundreds of games.

The Softserve API is available at <https://softserve.harding.edu>. Beyond this guide, technical documentation is available in two formats:

- <https://softserve.harding.edu/docs>
- <https://softserve.harding.edu/redoc>

When in conflict, the technical docs supercede this guide.

The Softserve project is open source and hosted at <https://github.com/harding-university/softserve>. Students are encouraged to watch the repo and create issues (which do not have to be bug reports—they can be questions, feature requests, or anything else).

Softserve's core logic for the semester's game is handled by an external engine, which is left unpublished until the end of the class. Technical output from this engine (most notably the version and build time) can often be found in the `log` field of API requests.

## State and Action Notation

A snapshot of a game at a specific moment in time is called a **state**. A discrete move made by a player that takes the game from one state to another is called an **action**.

In order to use the API, your project will need to be able to both read and write game states in the notation used by Softserve. It also will need to be able to output action notation.

The state and action notation is specific to the semester's game, and is described in a different document.

## The State API

Softserve provides a set of endpoints for browsing states and actions. These are useful for checking your understanding of the notation and for debugging issues (with your project, or with Softserve's engine).

For example, `/state/initial` returns the initial game state (i.e. the board set up for a new game), and `/state/{state}/actions` returns a list of actions available from a given state.

See the API docs for the `/state` endpoints for details.

The state API is for understanding, testing, and debugging. Projects should not rely upon it for core processing, and it is not guaranteed to be available during the official tournaments.

### Interactive UI

Softserve also provides an interactive UI for visualizing and navigating between states. This can be found at <https://softserve.harding.edu/ui/>.

The URL hash shows the current state, and can be set to any particular state you would like to visualize. Note that invalid notation produces undefined results.

### Creating a Player

Before you can participate in a tournament, you must create a player. To do this, post to the `/player/create` endpoint. Your request must contain a JSON object with two fields:

- `name`: The name of the player to create. This is up to you, and could be a team name, a code name, or a name for a specific version of your project. It must be unique.
- `email`: A contact email for the player, to be used if necessary by the Softserve admins.

Here is an example curl command to create a player:

```
curl https://softserve.harding.edu/player/create --json \
  '{"name": "example-player", "email": "example@example.com"}'
```

The request to `/player/create` returns a JSON object with a `token` field. This token must be stored, because it is used to authenticate the player on all subsequent API calls.

You can create as many players as you like. When organizing the official tournaments, the Softserve admins will request that each team submit a specific player name that will be used to identify and authenticate their project in the event.

### The AIvAI API

The core points of interaction between your project and the Softserve API are the `/aivai` endpoints. During a tournament, your project should follow this process:

1. Request a state to play from `/aivai/play-state`

2. (If the server returns HTTP 204, sleep a few seconds and go to step 1)
3. Read the returned state notation
4. Calculate an action to take using its AI algorithm
5. Post this action to `/aivai/submit-action`
6. Go to step 1

This should continue until manually terminated. See also the detailed process described below.

While all the student projects are following these steps, Softserve is tracking the (potentially hundreds of) games being played. When a call comes in from a player to `/aivai/play-state`, it finds a game with that player next to move and gives it to them. When they return an action, Softserve validates it and applies it to the state, and then when their opponent calls `/aivai/play-state` gives the game to them (although perhaps not immediately, if there are many games in progress).

This enables many games to be played in parallel, arbitrated by a third party, with each student project only needing to function as a “state in, action out” machine.

### Details

The request to `/aivai/play-state` requires a `player` and `token`, which can be obtained from `/player/create` (see above).

It also requires an `event`. This indicates which tournament you are participating in. The official class tournaments will have specific names.

`/aivai/play-state` returns a state and an `action_id`. The `action_id` must be saved and included in the corresponding call to `/aivai/submit-action`—it is how the server connects the two calls.

The server tracks how much time elapsed between the call to `/aivai/play-state` and the corresponding `/aivai/submit-action` call. If the time exceeds the instructor-specified thinking time, the game will be marked as a forfeit in the tournament results. (The game itself will continue to be played out.)

### Testing With the Mirror Event

A simple way to test your integration with the AIvAI endpoints is to set the `event` field to `mirror`. With that event, Softserve creates games where your player plays itself—that is, the state resulting from your `/aivai/submit-action` call is the state in the next `/aivai/play-state` call.

### Bugs

If you discover a bug in Softserve—and especially in Softserve’s core game logic—please report it. We are also here to help with any issues or questions you

have.

Either create an issue or email rschneid@harding.edu.