🧑‍🏫 Teacher Quickstart: Running a BotBattles Competition

This document describes how to run a BotBattles competition in a classroom or lab setting.

It is written for teachers who are:

- Comfortable working with technical tools
- Familiar with programming concepts (in any language)
- Interested in running a structured, repeatable bot competition

You do not need to be an expert in JavaScript or in BotBattles internals, but you should expect to engage with code, configuration files, and system behavior at a high level.

Overview: What You Are Running

BotBattles is a (nearly) **deterministic, tick-based simulation** in which autonomous bots compete in an arena.

Each bot consists of:

- A **build configuration** (`bot.json`)
- A **behavior implementation** (`behavior.js`)

During a match:

- The engine calls each bot's `tick(api)` function once per tick
- Bots observe, decide, and act independently
- The simulation runs without human input

As the facilitator, your role is to:

- Manage bot loading
- Run matches
- Interpret outcomes
- Support iteration

What You Need

- A modern browser (Chrome, Firefox, or equivalent)
- The BotBattles application loaded locally or hosted
- Access to student bot folders
- Time to run at least one test match before full competition

You should expect to:

- Read bot error messages
- Reload bots multiple times
- Observe and interpret simulation behavior

Bot Structure (Teacher-Level Summary)

Each bot is represented by a folder:

```
BotName/
├─ bot-*.json
└─ behavior.js
```

`bot.json`

Defines:

- Bot name
- Build tiers
- Wall behavior
- Optional cosmetic properties (e.g., color)

This file is validated on load.

`behavior.js`

Defines:

- A single function: `function tick(api) { ... }`
- The bot's decision-making logic

Runtime errors in this file affect only the bot that caused them.

`Eg. bot.json`

```
{
  "name": "WorstBot",
  "color": [255, 255, 255],
  "build": {
    "maxSpeedTier": 1,
    "turnRateTier": 1,
    "sightRangeTier": 1,
    "sightFovTier": 1,
    "shotPowerTier": 1,
    "shotRangeTier": 1,
    "maxHealthTier": 1,
    "memoryTier": 0,
    "wallBehavior": "stop"
  }
}
```

`Eg. behavior.js`

```
  function tick(api) {
    api.turn(10);
    api.fire();
  }
```

Loading Bots

1. Load the BotBattles website
2. Use **Load Bots / Select Bot Folders**
3. Select bot folders from your local filesystem
    a. either separately one at a time OR…
    b. All at once, if they are in a parent folder like:

```
MatchBots/
        ├─ UberBot/
        └─ LazyBot/
        └─ SniperBot/
```

4. The **Bot Load Report** section of the arena webpage will give you feedback on the success or failure of any bot loading attempts.

Interpreting the Bot Load Report

The report indicates:

- Successfully loaded bots
- Bots with schema errors
- Bots with behavior errors
- If a previous valid version was used (only happens if there was one)

Important characteristics of the loader:

- Bots are isolated from each other, correct bots will still load even if loaded together with a broken bot
- A broken bot does not crash the simulation
- Previously valid versions may be retained automatically

This allows for easier battle iteration for both students and teachers.

Running a Test Match

Before running a full competition, run at least one test match.

Recommended checks:

- Bots appear in the arena
- Bots move or act
- Errors are isolated and reported
- The simulation completes normally

Running Matches

Once bots are loaded:

1. Start the simulation
2. Allow it to run to completion
3. Observe overall behavior patterns and results
4. “Pause” and “step” to execute one tick at a time, if you want to highlight specific behavior OR, run the whole simulation to see how it unfolds

BotBattles is designed to be robust and fail elegantly if improper bot files are submitted.

Multiple Runs and Fairness

Single matches are not representative.

Reasons:

- Initial positions vary
- Some strategies are situational
- Small advantages compound over time

Recommended practice:

- Run multiple matches with the same bots
- Compare consistency rather than single wins
- Encourage students to justify changes based on repeated outcomes

Build Budgets

The build budget determines how many points bots can allocate across stats.

You may adjust the budget to:

- Increase design space
- Encourage specialization
- Allow more extreme tradeoffs

Higher budgets increase:

- Complexity
- Divergence between bots
- Difficulty of analysis

They do not guarantee better designs.

Common Issues and How to Handle Them

Bots fail to load

- Usually due to JSON syntax errors or schema violations
- Resolve text/code in files before rerunning matches

Bots crash during simulation

- Indicates runtime errors in `behavior.js`
- Affects only that bot
- Can be addressed between runs

Bots behave "unexpectedly"

- Often the result of:
  - Over-tuned stats
  - Over-aggressive turning
  - Insufficient scanning logic
- Treat as analysis opportunities, not bugs

What Is Expected Behavior

The following are normal and expected:

- Bots locking onto walls
- Bots missing shots frequently
- Bots dominating early and failing later
- Bots performing differently across matches

The system is intentionally transparent rather than "balanced."

Recommended Classroom Flow (Example)

Initial Session

- System overview
- Bot structure explanation
- First working bot
- Test match

Iteration Sessions

- Build tuning
- Behavior refinement
- Multiple match runs

Competition Session

- Repeated matches
- Aggregate results
- Post-match discussion

This structure is flexible and can be adapted to different classroom needs.

Final Notes

BotBattles is designed to:

- Expose the relationship between design choices and outcomes
- Make failures visible and recoverable
- Support experimentation and observational analysis

If bots load, the simulation runs, and results can be discussed meaningfully, the system is functioning as intended.