Science Olympiad Trial Event: Game Agent B (2026-2027)

Official Event Guidelines & Rules v:Sept.1 h1

1. Event Description & Overview

This event evaluates a two-member team's ability to design, build, test, and explain an original computer game paired with a functional Reinforcement Learning (RL) agent. This is a type of machine learning in which an agent learns through rewards and consequences. The project rewards modular software architecture, deliberate AI prompt engineering, isolated unit testing, and code literacy.

- **Team Size:** Maximum of 2 students.
- **Target Engine:** Godot 4.7.2 using GDScript.
- **RL Algorithm Constraints:** Simple Q-Learning
- **Game Type Options:** Side-Scroller (Obstacle Avoidance), Lunar Lander, Educational, or others if pre-approved by the Event Supervisor
- **Resources Link:** https://sciolygameagent.wordpress.com/

---

2. Project Lifecycle & Timeline

Phase 1: Digital Submission (Wednesday Night Before Competition)

Teams must digitally submit their entire Godot project directory and their separate maximum 4-Page Engineering Report. This gives the Event Supervisor time to review code mechanics, analyze documentation, and run code safety checks prior to Saturday's competition. Email the project link to w7eryron@gmail.com You will receive a reply indicating successful submission, or urge to retry.

Phase 2: Live Performance, Debugging Task & Interview (Saturday Competition Hour)

Teams will attend a scheduled 10-minute live demonstration window with the judge. Teams will be given a copy of their submitted program containing one syntax defect and one Unit integration defect. Teams will have a maximum of five minutes to identify, correct, and explain the defects. The defects will be designed so that a team familiar with its own program and the required five-Unit architecture should be able to identify and correct them within the allotted time. The game will have a clear/reset of the Q-table then execute a live training sequence of 2,000 episodes then run with the resulting learned table, and verbally defend their code logic and AI workflows.

Phase 3: The Afternoon Showcase (Saturday Public Viewing)

Following formal judging, all working student games will be loaded onto public workstations or a central projector. This provides student entertainment and interactive play during the afternoon scoring lull. Your game must be able to run in both human-player and AI Agent modes on the host computer, with specs of I5-class processor, 16gb Ram, integrated graphics, 1080p.

---

3. Game Interface & Code Structure Rules

Competition Interface & On-Screen Display Requirements

The competition interface must provide clearly accessible controls for Human Play, Fast Training, and Agent Evaluation. Additional controls are permitted only when they directly support testing or demonstration and do not substantially expand the project's scope.

Mandatory On-Screen Text Elements:

- Live Score Display: A text label or UI element that updates dynamically in real-time during both human play and agent gameplay.

- Game Status Banner: A clear visual indicator showing "WIN", "LOSS", or "GAME OVER", and a way to begin another play cycle, all at the conclusion of a play cycle.
- Mandatory introductory splash screen that explains game rules and keys.

The 5-Unit Modular Coding Requirement
Student game repositories must enforce a strict separation of concerns, dividing the core codebase into these five separate core scripts. Teams must implement the five required functional units as clearly identifiable and substantially independent modules. Additional supporting scripts are permitted but may not duplicate or obscure the responsibilities of the five required units.

- **Unit 1 (Core Game Logic):** Manages the boundaries, physics limits, obstacle movement, and the math function that converts complex continuous values into discrete coordinate integer groups (e.g., a `Vector2i` or `Vector3i` state space). This script alone determines the environment reward dictionary and tracks the live score and win/loss state.
- **Unit 2 (Human Input Controller):** Captures physical key commands and forwards basic, unweighted movement requests to Unit 1. It holds zero game rules or RL logic.
- **Unit 3 (Agent Brain):** Houses the pure math engine of the Q-table dictionary. It takes an integer state input, calculates action choices using an Epsilon-Greedy strategy, and runs the mathematical adjustment algorithm.
- **Unit 4 (The Integration Loop):** Acts as the system coordinator. It automates the environment reset sequence, counts total episodes, passes states between Unit 1 and Unit 3, and controls the training loop execution speed.
- **Unit 5 (Testing Harness):** Contains the code used to independently test the required units and to perform system-level integration tests.

---

4. The 4-Page Technical Design Report ("TDR") Requirements

In place of in-game documentation screens, teams must submit a maximum 4-page document (minimum 10-point standard font, single-spaced) tracking their development processes:

1. **Unit Separation & Strategy Log:** A brief technical map explaining how they broke their project down into the four required scripts and verified their isolation.
2. **AI Prompting & Debug Checklist:** A selection of representative prompts, reproduced exactly, sent to their generative AI assistants (e.g., Gemini, Copilot). It must also highlight at least one major logical error, syntax problem, or Godot 3 vs. Godot 4 version mismatch introduced by the AI, detailing how the students modified their prompting or code to resolve it.
3. **Hyperparameter Optimization History:** A text log tracking their changes to Alpha (learning rate), Gamma (discount factor), and Epsilon (exploration decay). They must state how changing these values impacted the total episodes required to achieve stable human-level performance.
4. **Human Performance Benchmark:** Teams establish and document a representative human-player performance benchmark for their game. The benchmark should describe the test method and results and will be used for comparison with the trained Agent.

5. Scoring

Total 100 points. See the Scoring Guide Matrix on the website.
Score = VPF (/15) + AICH (/20) + TDR (/20) + RLMAP (/25) + LI (/20)
The first tiebreaker is the score in Section 5 Live Interview, second is the score for Section 4 RLMAP.
See the Q&A on the website for more details.

Scoring Guide Matrix (100 Points Max)

Scores are calculated using the pre-submitted files combined with the live 10-minute interview.

Section 1: Visual Presentation & Input Framework (15 Points)

- **Main Menu Functionality (5 pts):** Screen architecture successfully transitions between splash screen, human play, background training, and agent evaluation without freezing or generating Godot script errors. Includes functioning Live Score and Win/Loss displays.
- **Human Usability & Responsiveness (5 pts):** Keyboard or mouse controls map correctly, permitting a human player to interact cleanly with Unit 1's structures.
- **Game Playability and Interest (5 pts):** The Game is interesting and fun to play.

Section 2: Architectural Integrity & Code Hygiene (20 Points)

- **Modular Separation of Concerns (10 pts):** Scripts follow the 5-Unit rule. The RL brain (Unit 3) has no direct references to display graphics, and the Core Game Logic (Unit 1) contains no input handling. Signals and get_node strings are allowed. System testing is handled in Testing Harness (Unit 5), with necessary links into code in other Units.
- **Logic Formatting & Readability (10 pts):** Node configurations follow clear naming hierarchies. Scripts feature custom comments explaining functions and preventing variable scope leakage.

Section 3: TDR Documentation & AI Workflow (20 Points)

- **Isolated Unit Testing Records (10 pts):** Report presents explicit proof that units were tested independently (using console print check verifications) before being wired together.
- **Prompt Engineering & AI Error Correction Log (10 pts):** Report includes representative prompts reproduced exactly and documents active debugging of AI errors, proving students controlled the AI tool rather than letting it dictate design blindly.

Section 4: RL Math & Agent Performance (25 Points)

- **Discretization Integrity (10 pts):** Game logic cleanly transforms continuous coordinates or velocities into a strict integer index vector (`Vector2i` or `Vector3i`) without leaking loose decimals into the Q-table key dictionary.
- **Reward Balance Strategy (10 pts):** Step penalties and success/failure values successfully force convergence, steering the agent toward efficient pathways without getting stuck in infinite safe loops.
- **Demonstrated Agent Performance (5 pts):** The trained Agent demonstrates consistent, effective progress toward completing the game's objective.

Section 5: Live Interview (20 Points)

- **Verbal Code Literacy (10 pts):** When pointed directly to a selected code statement or defect in their script, students accurately explain its logical role, proving code ownership.
- **Live Training & Performance Validation (10 pts):** The agent runs its training episodes cleanly within the allotted timeframe, changes its Q-values noticeably, and achieves a win-rate matching or beating the team's documented human performance benchmark.

Logistics Addendum and Q&A

Competition Conditions & Standardization Q&A

1. **Internet access:** Is it permitted during development at the competition? Answer: definitely yes. During the interview or competition hour? - Answer: No.
2. **AI access:** Only during project development … not during the competition hour.
3. **External assets:** Downloaded sprites, sound effects, and code libraries are permitted. Cite sources in your TDR.
4. **Project Size:** The compressed project should be reasonably sized for digital transfer. Project files must be cleaned (deleting the `.godot` cache folder), zipped, and renamed to match the required header format before being emailed as an attachment. Cloud links are only permitted as an emergency fallback if the file exceeds email limits. Include your TDR in this ZIP file.
5. **Starting state:** Empty Q-table, pre-trained Q-table? Your submission for scoring should include both your optimized pre-trained Q-table, and the option to clear it.
6. **Randomness:** Must teams use a random seed supplied by the judge? No.
7. **What does "clearing the Q-table" mean:** Clearing the Q-table means returning the Agent's learned action values to its initial untrained state.
8. **Host computer setup:** Who supplies Godot, and which exact version is installed? Godot 4.7.2 will be used on the host computer, provided by the Event Supervisor.
9. **Code Safety Isolation:** Any project containing scripts that attempt to access files outside the project directory, execute terminal/system commands (`OS.execute`), or download external network dependencies will be instantly disqualified for safety reasons.
10. **Live Training Convergence:** What happens if my Agent training is run after the Q-table is cleared during the live demonstration, and my training does not converge? Assuming that the training logic looks to be correctly functioning, the observable Q-values appear to be learning, and state/reward design appears reasonable, then the judge will try it again, and if needed, try it on the team's computer. If it still does not converge then there will be a slightly lower score for performance.
11. **Project dependencies:** Any dependencies must be included with the submission. All code is to be written using only Godot 4.7+. Projects must be self-contained and may not require external runtime libraries, plugins, or software beyond the specified Godot installation.
12. **Submission format:** ZIP file, cloud link, repository, or other standardized method? - Answer: see the required submission format, and recommended procedure given below.
13. **Failure contingency:** What happens if a project runs on the team's machine but not the host machine? - Answer: You will be allowed to demonstrate your game on your computer, but you will have a 10% deduction applied to your final score, and it won't be shown in Phase 3. Note that this usually happens if you code includes hardcoded paths, use only absolute paths.
14. **Accessibility and hardware:** Are personal laptops permitted during the live demonstration? - Answer: Yes. Personal laptops may be used for demonstration if authorized by the Event Supervisor. Projects are normally scored on the host machine; see the Failure Contingency rule for projects that do not run there.
15. This Q&A will be expanded as needed on the supporting website at: https://sciolygameagent.wordpress.com/

Game Agent Godot Project Saving & Submission Checklist

Follow these 3 steps to save your game properly to submit it for scoring.

1. Close and Clean Up

- [ ] **Save your scene** inside Godot (`Ctrl + S` or `Cmd + S`).
- [ ] **Close the Godot Engine completely** (closing the app prevents locked files).
- [ ] Open your project folder and delete the folder named **`.godot`** (this is just a temporary cache; deleting it makes your file VERY tiny and fast to upload).

2. Zip and Rename Your Folder (Choose your OS)

- [ ] **Windows 11:** Right-click the folder ➔ **Compress to ZIP file**.
- [ ] **Windows 10:** Right-click the folder ➔ **Send to** ➔ **Compressed (zipped) folder**.
- [ ] **Mac:** Right-click (or Two-finger tap) the folder ➔ **Compress [Folder Name]**.
- [ ] **Linux:** Right-click the folder ➔ **Compress...** ➔ Select **.zip** format.
- [ ] **Chromebook:** Open the **Files** app ➔ Right-click (or Two-finger tap) the folder ➔ **Zip selection**.
- [ ] **Rename the ZIP file for scoring submission to the Event Supervisor:** Change the name to match this exact header format:
  `GameAgent2027.<yourSchool>.<yourTeamNumber>.<yourFirstNames>.zip`

3. Save to Cloud & Email to Event Supervisor

- [ ] **For Work at Home:** You are welcome to use Git or similar if your school system allows. If not, then you can upload your ZIP file to your personal cloud storage (Google Drive, OneDrive, etc.) so you can download and extract it at home.
- [ ] **For Scoring:** Open your email and send your project ZIP file as an attachment to: **w7eryron@gmail.com**
    - [ ] *Requirement:* Use that same exact header format as your email subject line.
    - [ ] *Requirement*: if for any reason your ZIP file cannot be sent, send an email explaining what happened, and a link to your file with that same header.

Note that if you are comfortable using Git for your project development, it is OK, and frankly, encouraged, to continue using it, and to send the zipped file of the commit that you want scored. Don't merely send the link to your Github account and expect the Event Supervisor to download it - you are not permitted to work on your game after the due date.

2026-2027 Trial Event: Game Agent Coach's Survival Guide

Dear Science Olympiad Coach,

If you look at the words "Reinforcement Learning," "GDScript," or "Matrix Discretization" and feel a wave of panic, take a deep breath! **You do not need to be a computer scientist, a mathematician, or a video game developer to lead a gold-medal team this season.**

In fact, teams with coaches who do not know how to code often perform better. Why? Because instead of writing the code for the students, you will act as an **Engineering Project Manager**. Your job is to guide their schedule, monitor their design logs, and ask the right questions.

The Secret: AI is the Programming Assistant

This event explicitly encourages the use of Generative AI tools (like Gemini, ChatGPT or Copilot). The AI handles the syntax, commas, and heavy lifting. Your students are responsible for the architecture, testing, debugging, mathematics, and final code. They break the game down into instructions, feed them to the AI, test the outputs, and optimize the Q-learning math dials.

Your Strategic Master Checklist

To keep your team on track for a top-tier finish, use these milestones during your team practices:

- **[ ] Phase 1: Enforce the 5-Unit Rule.** Ensure your students do not let the AI write one giant script. Check their project folder. They must have five distinctly separated scripts: Unit 1 (Rules), Unit 2 (Keyboard Inputs), Unit 3 (Agent Brain), Unit 4 (The Coordinator Loop), and Unit 5 (Testing Harness). .
- **[ ] Phase 2: Check for Isolation Testing.** Ask your students: *"Have you run Unit 1 by itself yet?"* They should be able to show you plain text messages printing to the console screen verifying scores before they even connect arrow keys or graphics.
- **[ ] Phase 3: Benchmark the Humans.** Long before tournament week, make sure your students record their own game scores. They need a baseline human performance benchmark so they can determine whether their trained Agent matches or exceeds human performance. .
- **[ ] Phase 4: Track the AI Prompt Transcripts.** Remind students to save their AI conversation logs. If the AI outputs an error or breaks, they shouldn't panic—they just need to document the mistake and show how they corrected their prompt to fix it.

High-Utility Questions to Ask Your Team

When checking in on your team's progress, use these phrases to test their understanding:

1. *"If we want to make our obstacles move faster, which of your 5 units handles that?"* (Ans: Unit 1).
2. *"Show me the script where the AI assistant made a mistake. How did you change your prompt instructions to steer it back on track?"*.
3. *"Is your Q-Table storing clean, whole numbers, or are raw pixel decimals leaking in?"*.

Thank you for volunteering your time to coach these students. You are helping them master the fundamentals of system design, automation engineering, and responsible tool use!

- Ron Wright, Event Supervisor w7eryron@gmail.com, 360-953-9476

Draft Rubric from Scoring Guide - (v.h this will likely be modified after each tournament)

Section 1: Visual Presentation & Input Framework (Max 15 Points)

- **13-15 Points:** Seamless transitions between Splash Screen, Human Play, Fast Training, and Agent Evaluation. On-screen displays (Live Score, Win/Loss/Game Over Status) update instantly without errors. Game is highly responsive, fun, and playable.
- **10-12 Points:** All modes function, but transitions might experience slight visual stutters. UI displays are accurate but layout or font sizing is slightly awkward. Controls work cleanly.
- **5-9 Points:** One mode fails to run (e.g., Fast Training crashes, but Human Play works). UI elements overlap or fail to clear at the end of a cycle. Controls feel clunky.
- **1-4 Points:** Game crashes frequently during navigation. Critical UI indicators (Score/Status) are missing or broken. Game is unplayable.

Section 2: Architectural Integrity & Code Hygiene (Max 20 Points)

- **17-20 Points:** Flawless 5-Unit separation. Unit 3 has zero references to graphics; Unit 1 handles no user input. Custom comments explicitly prevent variable scope leakage. Node naming is uniform.
- **13-16 Points:** 5-Unit architecture is followed, but an isolated graphic reference exists in Unit 3, or a stray input catch is left in Unit 1. Nodes use clear names, but comments are sparse.
- **7-12 Points:** Code is lumped into 3 or 4 scripts instead of 5. Scope variables leak between units. Hardcoded asset paths (C:/...) are present instead of relative paths (res://).
- **1-6 Points:** Monolithic "spaghetti code" structure with little to no separation of concerns. Node naming is generic (Node2D, Button2), and no descriptive comments exist.

Section 3: TDR Documentation & AI Workflow (Max 20 Points)

- **17-20 Points:** Technical Design Report provides explicit proof (console print verify logs) of independent unit testing. AI log reproduces exact prompts, highlighting a major error resolution where students maintained absolute control.
- **13-16 Points:** Report documents unit testing, but lacks console verification logs. AI prompt logs are present but focus on minor typos rather than deep logical/version debugging.
- **7-12 Points:** TDR is missing entire sections (e.g., skipped the AI log or the Testing Log). Document formatting violates the font/spacing guidelines.
- **1-6 Points:** Highly disorganized, brief, or generic report with no specific prompts, console text, or verification logs provided.

Section 4: RL Math & Agent Performance (Max 25 Points)

- **21-25 Points:** Continuous values cleanly discretize into integer spaces (Vector2i/Vector3i) without leaking decimals. Reward strategy forces swift convergence without safe loops. Agent visibly outperforms or matches the human baseline.
- **15-20 Points:** Discretization works but is overly complex. Reward balance converges, but the agent takes a high number of episodes to learn due to weak step penalties. Agent matches human performance.
- **8-14 Points:** Loose decimals occasionally slip into the Q-table key dictionary. Reward system allows the agent to get stuck in infinite loops, or training fails to reach convergence after 2,000 episodes.
- **1-7 Points:** No functional discretization (agent relies on continuous values). Reward dictionary triggers errors. Agent moves entirely at random with no signs of learning.

Section 5: Live Interview (Max 20 Points)

- **17-20 Points:** Both team members accurately explain selected lines of code instantly, proving total ownership. Live training executes cleanly, the Q-table demonstrates measurable changes during training , and the agent matches/beats the human benchmark within the 10-minute window.
- **13-16 Points:** Students explain code logic well but struggle with technical terms. Live training functions and shows learning, but the final win-rate falls just short of the human benchmark.
- **7-12 Points:** Only one team member can answer code questions, revealing an un-balanced workload. The agent runs its training episodes but its Q-table changes are stagnant or chaotic.
- **1-6 Points:** Neither student can explain basic script operations, indicating total reliance on generative AI without comprehension. Training loop crashes or freezes during the live test.

Science Olympiad Game Agent B: AI Prompting Cheat Sheet

You are encouraged to use these structured prompt rules whenever you talk to Gemini, Copilot, ChatGPT or your chosen AI coding assistant. . Your Technical Design Report (TDR) requires you to log these prompt transcripts.

Rule 1: Set Up the AI's Persona (Your First Prompt)

Never ask an AI for code without telling it your specific rules first. Paste this exactly as your first message in a new chat:

> *"You are an expert Godot 4.7+ programmer using GDScript. We are building a modular game for the Science Olympiad using strict separation of concerns. You must never combine physics, user input, and AI math into one script. Always write code targeting a 5-Unit architecture: Unit 1 (Core Game Logic), Unit 2 (Human Input), Unit 3 (Agent Brain), Unit 4 (Integration Loop), and Unit 5 (Testing Harness). Do you understand this 5-Unit layout constraint? Please see the attached Event Description and notes. "* And of course, attach this entire 8-page pdf document if your system allows it.

Rule 2: Use the "Isolation Guard" Template

When asking for help with a specific script, lock the AI into that exact module using this format:

> *"I need help writing a function for **[Insert Unit Name here, e.g., Unit 3 (Agent Brain)]**.*
>
> *Remember the architectural constraints: This script is only allowed to **[Insert rule here, e.g., run the Epsilon-Greedy strategy math]**. It cannot handle keyboard inputs or touch display graphics.*
>
> *Here is my current code snippet: **[Paste your code here]**"*

Rule 3: How to Feed Errors to the AI

If Godot shows an error banner in the debugger console, do not panic! Copy it, paste it into the AI, and use this template:

> *"My Godot 4.7 debugger threw this error: **[Paste exact error text here]**.*
>
> *This happened inside **[Insert Unit Name]**. Fix the bug, but do not combine this file with other units or break our modular boundaries."*

TDR Log Requirement Reminder:

Every time the AI gives you a broken script, a syntax error, or a Godot 3 vs. Godot 4 version mismatch, you **must** take a screenshot or copy the prompt history. Log the exact prompt you used to correct the AI and steer it back on track for your TDR Section 2.

PS - you may try to ask your AI coding assistant to write your entire game for you in one shot, it probably won't work, but try it if you want to, anyway - be sure to tell it what kind of game you want, … and, if you do, you will need to do **a lot** of work to correct errors and to understand what your AI did. Good Luck !

Testing Harness — Student Guide

The **Testing Harness** is a student-designed testing system used to verify that the required software units work.

Testing should happen in two stages:

1. Independent Unit Testing: Test each unit by itself before combining the units.

2. Integration Testing: After the individual units work, test the units together as a complete system.

The Testing Harness may contain additional helper functions or test code. It should **not** contain the game logic, player-input logic, or reinforcement-learning algorithms being tested. The Testing Harness will need to communicate with the other Units by calling functions in those Units and checking their results.

Unit Testing

Do you know what you are doing? :
During practice, pick any function in their script, delete a single closing parenthesis `)` or colon `:`, and ask your partner to fix it. If they know how to read the error console and replace it instantly without opening ChatGPT, they pass the literacy check!

Before integration, ask:

Unit	Question to Test
Unit 1 — Environment	Does the environment behave correctly?
Unit 2 — Actions	Do inputs produce the correct actions?
Unit 3 — Agent Brain	Does the Q-learning mathematics work correctly?
Unit 4 — Integration Loop	Does the training sequence correctly connect the environment, actions, and Agent Brain?

A good unit test should produce a result that you can observe and verify. Don't just ask whether the program runs — ask whether it produces the **expected result**.

Integration Testing

Once the four units have passed their individual tests, test them together.

Step 1 — Communication

Do all four units communicate correctly? Can you verify that the information passed between Units is correct?

Check that:

- the environment provides the state;
- the agent selects an action;
- the environment returns a result and reward;
- the agent updates its Q-values;
- the training loop repeats correctly.

What if it's not winning?
You may have too many states to track. Instead of tracking pixel position `432`, divide the coordinate by a grid block size (like `64`) and round to a whole number using integer division: `432 / 64 = 6`. Now the agent only has to remember state `6` instead of hundreds of individual pixels!

Step 2 — Learning

Does the complete system actually learn?

Run enough training episodes to determine whether the agent begins to discover successful paths.

Look for evidence such as:

- successful training episodes;
- increasing or improving performance;
- a developing Q-table;
- a repeatable path to the goal.

The Secret to Rewards: The "Step Penalty"
If you want your agent to reach the right side of the screen quickly, you must penalize it slightly for wasting time! Give it a tiny negative reward (like `-0.1`) for every frame or step it takes without reaching a flower or the goal line. This forces the agent to learn the fastest path instead of sitting still.

Step 3 — Evaluation

Does the learned policy work without exploration?

Set: **ε (epsilon) = 0**

The agent must now use its learned policy rather than randomly exploring.

Run the agent several times and record the results.

Ask: **Does the trained agent consistently find a successful path?**

Testing Beyond "It Works"

Once the integrated system works, begin asking questions like these.

- How much training is needed?
- Does changing a training parameter affect performance?
- Does the starting random seed affect the result?
- Does the agent reliably find a successful path?
- Can the result be reproduced?
- What happens when we change **one variable at a time**?

These are experiments, not simply debugging tests. This is the process of scientific investigation.

Conjecture → Test → Record → Compare → Explain

Competition Debugging Task

The Testing Harness is designed to help you prove that your Units work correctly. At competition, you will also be asked to demonstrate that you understand how those Units communicate.

You will be given a copy of your submitted program containing:

- **one syntax defect**, and
- **one Unit integration defect**.

You will have **five minutes** to identify, correct, and explain the defects.

The integration defect will involve communication between required Units rather than an arbitrary change to your game's design. The goal is to determine whether you understand how information and commands move through your five-Unit system.

Practice: Be prepared to trace information through your program:

Environment → State → Agent → Action → Environment → Reward/Next State → Agent