Integrating Machine Learning with an FPS Aim Trainer for Optimal Sensitivity Finding

By: Sharan Krishna

Computer Science Department
California Polytechnic State University, San Luis Obispo

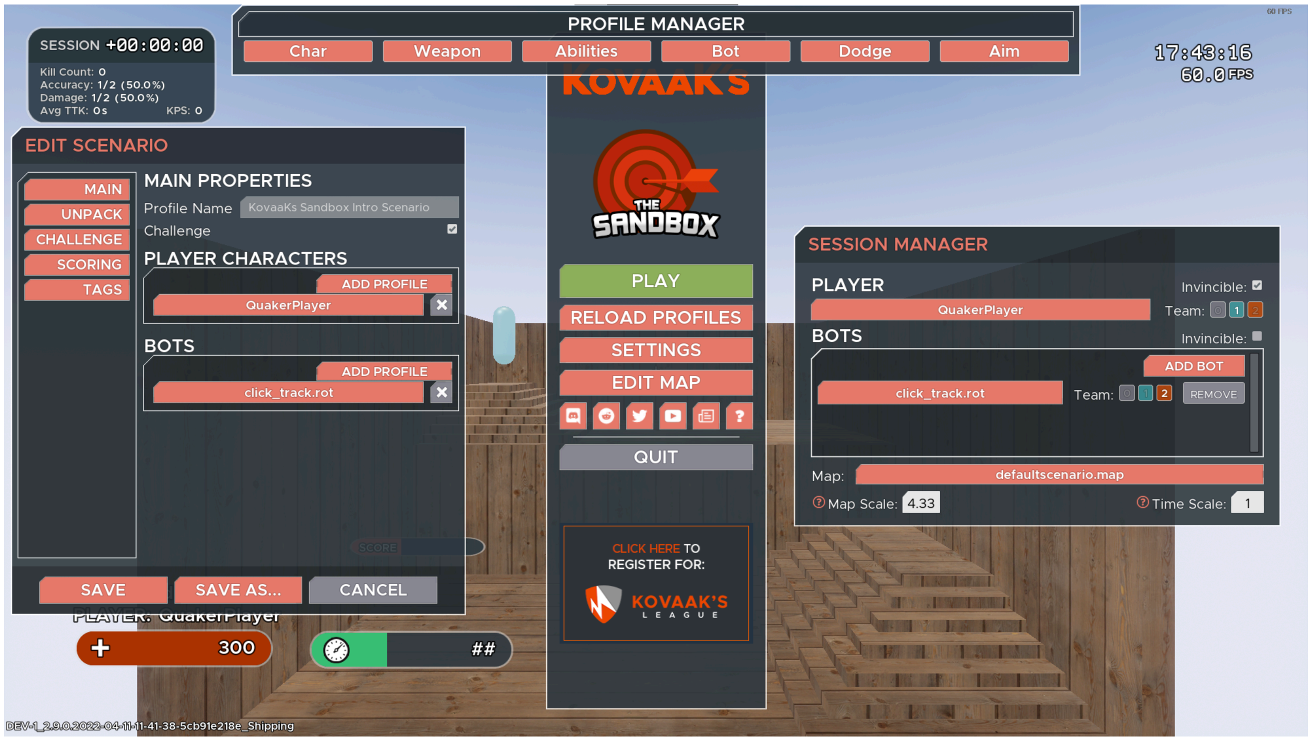
December 2025

Introduction

First-person shooter (FPS) games often demand high levels of skill in aiming, which leads players to look for external tools to improve their performance. This is where the concepts of aim training and aim trainers come in, becoming an easily accessible outside source for players to strengthen their performance with custom scenarios outside a set game. While many aim trainers exist, they offer limited insight into player performance metrics or adaptability to varying aiming styles. Furthermore, most existing aim trainers lack a standardized way of correlating aim skill with real-world performance or personalized feedback. This aim trainer addresses these limitations by providing a modular training environment that includes multiple aiming tasks with real-time feedback, a machine learning integration to adapt to player weaknesses, and precise scoring and accuracy systems. The goal in this project and the aim trainer is to both train reflexes and tailor the training experience to each player, adjusting their sensitivity in real time to follow their performance and help them find an optimal number to play with.

Background / Related Work

Popular aim trainers like Kovaak's FPS Aim Trainer and Aim Lab have been widely used by competitive FPS players. These tools offer structured drills and data tracking, but they are either heavily focused on cosmetic gamification or lock their customization features behind paywalls. Aim Lab, for example, contains performance analytics, but doesn't provide pure improvement statistics and sticks most real features behind paywalls and gamification. It also features a sensitivity finder to give the illusion of a personalized experience, but it is also hidden behind a paywall. Kovaak's, meanwhile, has no gamification, but attempts to cram a lot of content into one space. This makes it too complicated for inexperienced users to hop in and play, and makes the more fun features like task creation look even more daunting for the limited personalization that it already offers. This is shown in Figure 1, where the task creation system is covered in text and complex features that would take a new user many 3rd-party tutorials and research to understand and achieve even a simple level of customization. This aim trainer differentiates itself by prioritizing open-ended modularity and extensibility, having a simple UI and detailed graphs for users to view their progress and understand what they are doing every step of the way, without being hindered by complex features or gamified pop-ups and advertisements.



SESSION +00:00:00
Kill Count: 0
Accuracy: 1/2 (50.0%)
Damage: 1/2 (50.0%)
Avg TTK: 0s KPS: 0

17:43:16
60.0 FPS

MAIN PROPERTIES
Profile Name: KovaaKs Sandbox Intro Scenario
Challenge: ✓
PLAYER CHARACTERS
QuakerPlayer
BOTS
click_track.rot

PLAYER Invincible: ✓
QuakerPlayer Team: 1
BOTS Invincible:
click_track.rot Team: 2
Map: defaultscenario.map
Map Scale: 4.33 Time Scale: 1

PLAYER: QuakerPlayer
300
##

DEV-1_2.9.0.2022-04-11-11-41-38-5cb91e218e_Shipping

Figure 1: From Kovaak's Wiki: "Intro to Scenario Creation".

Design

As previously mentioned, the project is structured around the idea of modularity and extensibility. Each aiming task is built in a separate scene with reusable architecture and scoring logic. The tasks are sorted into 4 categories: Tracking, Flicking, Speed, and Precision. The tasks themselves vary over a wide range of options, some being similar and some being different.

The basic tracking tasks are variations of a basic task, where a sphere moves across the screen while the user must track it by holding down the left mouse button. The main goal here is to track accuracy, since the more accurate a user is at keeping their crosshair on the target, the more closely aligned their sensitivity is to their own movement. There are three variations on this task, including one with random speed, diagonal movements, and 8-direction movement to cover all possible tracking variations in a first-person shooter. The final tracking task is named EntryTrack, where a sphere moves in one direction, and the user must click on the sphere to force it to move in the other direction. Upon hitting it 3 times, a new sphere appears. Here, we track accuracy as the number of hits vs. misses, testing both the user's tracking and ability to click at the correct time.

For flicking tasks, the main one is Gridshot, where three static targets appear in a grid, rewarding fast and accurate clicking. Here, we track accuracy as the number of hits vs misses; the more accurate the player is, the better the sensitivity is for them. There are many more varying tasks, including headshot, where one sphere appears somewhere on the screen and, upon being hit, moves to the other side of the screen at the same Y level. This is meant to help the user focus on keeping the crosshair at head height while focusing on flicks. Then, there is Spidershot, where one target appears in the center of the screen and, upon being hit, it moves somewhere to the left or right, then back to the center, then to the other side, and repeats. This is meant to focus on keeping the crosshair at head height while focusing on flicks. Finally, there are Flickshot and Single Flickshot, where two or one targets spawn somewhere and need to be flicked to as a pure flick training exercise. The different number of targets in the two tasks is meant to teach target focus, since having multiple can disrupt a flick.

For speed tasks, the main goal is to test how many targets you can get in a certain period of time while still being accurate. The first one is Kinggg, named after the pro Valorant player of the same name. Six targets appear on the screen, and the user must both precisely and quickly go from target to target, shooting them all. Next is Multishot, where targets appear randomly on the screen, growing in size before disappearing altogether. The user must hit the targets as quickly as possible before they disappear to increase their score. Finally, there is Zapshot, where a target appears briefly and then disappears, making the user react and flick to hit before it's gone. Instead of measuring accuracy here, we measure reaction time to flick and hit the target in milliseconds. Both the best and average RT are saved.

Finally, we have precision tasks. The first is the PrecisionCircle task, where a sphere moves in an orbit, sometimes covering an "ideal zone." Players are scored based on how often they hit on this ideal zone, with the sphere speeding up its orbit every time they hit. This does not affect the sensitivity portion of the aim training, but is for the user to practice simple shot timings and patience, which is key in first-person shooters. Next is sixshot, where six targets spawn very far away, and the user must move between them and hit them while being precise. Finally, there are two moving target tasks, where a target moves left and right, and the user must both track and shoot it before a new moving target spawns in. This comes in two tasks, one with a fixed speed and one with the targets having changing and random speeds. These tasks track accuracy, as the user is meant to be precise in their aim to improve.

Each task shares a consistent UI/UX style and sensitivity management from a singleton design for the model to update it as the tasks are played. The design emphasizes task-specific performance metrics to support the machine learning feedback and sensitivity tracking through a visual graph shown in the task-select menu.

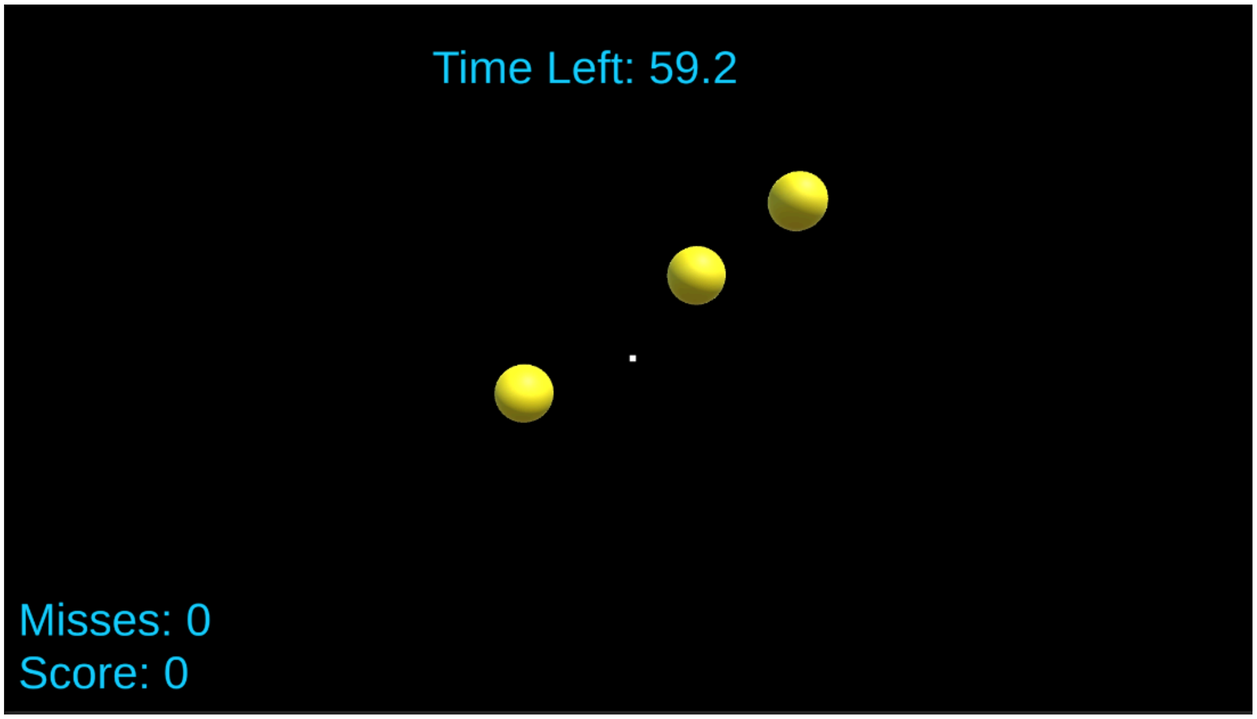


Figure 2: The basic Gridshot task

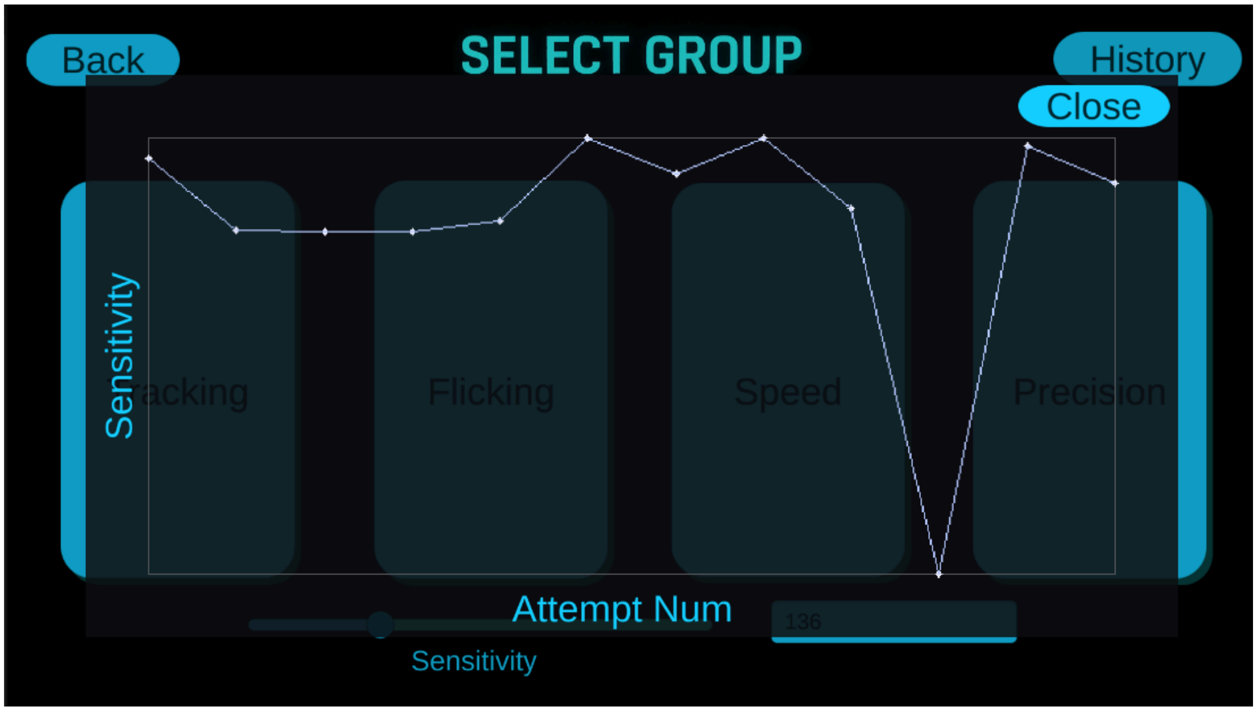


Figure 3: The Sensitivity graph for the user to track their change for different attempts

Implementation

I chose to implement the project in Unity for its extensive support for rapid UI development and modular scene management, and for familiarity due to using it for CSC 378. The core components include the SensitivityManager, created as a Singleton with centralized control of mouse sensitivity across all tasks, the scoring scripts for each task, procedural movement for different tracking and other tasks, and the core machine learning model that feeds in the new sensitivity for the player.

Creating individual tasks was surprisingly simple due to Unity's component-based workflow. Most tasks were built by writing short C# scripts that spawned targets inside a defined bounding area, moved them in a particular pattern, and responded to player interaction through raycasts or continuous tracking checks. Each movement behavior, like strafing, sinusoidal motion, or randomized speed, was just a small script that updated the target's position every frame. Hits and misses were handled through a consistent event-based system, and every task had its own lightweight manager script responsible for scoring and UI updates. This modular structure made it easy to create new tasks quickly by mixing and matching different spawning, movement, and scoring behaviors.

The most complex implementation was the actual AI portion, which contained two levels: a per-task 1D Bayesian Optimization with a real Gaussian Process for continuous sensitivity exploration and a Contextual layer that shares learning across tasks using a lightweight Bayesian Linear model, or Thompson Sampling. The system logs a compact set of metrics after every completed run, including the sensitivity used, task-specific metrics like accuracy or flick error, and contextual information like FOV or target distance to automatically adapt mouse sensitivity based on player performance. These data are compressed into a single utility value between -1 and 1, which reflects how well the run aligned with the desired performance profile for the task. Each task maintains a short rolling history of recent runs, stored as a simple JSON in Unity's PlayerPrefs. On top of this data, the core learning algorithm uses a one-dimensional Gaussian Process with an RBF kernel to model the relationship between sensitivity and performance, and an Expected Improvement acquisition function to recommend the next sensitivity value. A small trust region limits how much sensitivity can change between runs, just to ensure stability and prevent extreme jumps. During the early runs, the system performs structured exploration by sampling a set of candidate sensitivities around the user's starting value, after which the model relies on the Gaussian Process's uncertainty estimates to balance exploration and exploitation. The system can optionally incorporate a contextual prior using Bayesian linear regression to help predict a likely "good" sensitivity range before optimization begins. While complicated, this approach provides a lightweight and robust optimization loop that continuously personalizes sensitivity in response to user behavior, while remaining computationally inexpensive and fully local to the game.

Unity UI elements like TMP_Text were used to present scores, accuracy, and feedback in real-time. Additionally, Unity's simple PlayerPrefs allowed easy feeding of calculated values to the Sensitivity Singleton to update values and the player's current sensitivity while also displaying all previous values for a player on a graph.

Analysis / Verification

The primary goal of this project was to build a functional, modular aim-training application with machine learning capabilities that adjust mouse sensitivity based on user performance. I defined success by two key criteria: if players found the interaction intuitive and engaging, and if the lightweight ML system meaningfully adapted sensitivity over play sessions.

The machine learning system in its current state proved functionally correct: it successfully tracked per-task performance metrics, stored histories, and updated sensitivity values between runs. Although the adjustments were intentionally conservative, playtesters consistently confirmed that sensitivity values did update and that the change made sense relative to their performance. This verifies that the ML pipeline functions end-to-end and can be replaced with even more advanced optimizers in future work, perhaps using more data to train a multi-dimensional neural network.

I conducted playtesting with about 15 individuals for both versions, each providing multi-point feedback on UI clarity, feel, learning curve, and overall usability. Several themes emerged. Players consistently described the tasks as responsive and "satisfying" in terms of hit feedback and target behavior, with the modular task structure allowing them to quickly understand each mode without additional instruction. The simple addition of audio feedback for hit sounds and UI clicks definitely increased the sense of polish and game feel, as taught in CSC 378. The further additions of the pause menu and options for sensitivity graphs improved usability and made the experience feel more like a complete game. The final consistent positive was the model allowing for gradual sensitivity adjustment, eventually leading to a consistent final sensitivity that the users tend to like.

Before the 100% playtest, several users found the UI visually plain, with the menus lacking a strong identity. They also wanted some visualization of their progress and how things were changing in between tasks, since the original model was too subtle. Players expected greater changes after noticeably poor performances. These were all updated by the 100% playtest, with a fully revamped UI and color scheme, complete with graphs to map sensitivity and a fully working new model.

I added fixes according to these comments throughout development. For example, audio feedback was redesigned to provide continuous tracking sounds, ML scaling parameters were adjusted, the pause menu was redesigned for consistency, and several small UI/UX fixes, like cursor locking and button responsiveness, were applied.

Overall, the project succeeded in delivering a complete aim-training environment with polished flicking, tracking, and reaction-time tasks, a persistent global sensitivity system, a working performance-driven ML loop, graphical overlays that visualize sensitivity changes over time, and a modular architecture capable of supporting more advanced ML models

Future Work

There a a few features I want to add, given more time. The current machine learning model, does not yet use a full Bayesian optimization pipeline and instead adopts a lightweight version, but the current system verifies that the architecture is capable of supporting it. The project demonstrated scalability (tasks, ML bridges, and scorers can be added easily), efficiency (all scenes maintain high performance), and fault tolerance (no major crashes or fatal errors during testing). The interface, while visually simple, remained intuitive enough for playtesters to understand without instruction. Another thing I wanted to add was a calculator to convert the Unity Sensitivity value into a direct game usable value, but the Unity values are arbitrary and would have to be calculated through math on how much movement is done per mouse swipe, given additional metrics like DPI. This is implementable and could easily be done in the future with some looking in the backend.

Conclusion

This project has managed to provide a fully functional aim training environment that is modular and supports a wide range of tasks such as flick tasks, speed tasks, and tracking tasks. This has been achieved with the help of a unified task structure and a scoring system. The application provides users with instant feedback on their performance in all tasks, with the most prominent enhancement being the fully functional machine learning pipeline. Each task now streams standardized performance metrics into a persistent data-logging system. These data points feed a sensitivity-optimization model that adapts the player's mouse sensitivity over time. Unlike commercial trainers that rely solely on static player settings or gamified progression, this system actively learns from user performance and adjusts a core control parameter to support a more tailored and efficient training experience.

Although this version uses a lightweight optimization model, it is designed to be fully replaceable with more advanced approaches such as Bayesian Optimization or contextual multi-armed bandits. The key infrastructure, with data collection, per-task bridges, persistent historical logs, and visualization tools, is already in place. As a result, the project stands on solid ground for future personalization features, such as automated difficulty scaling, per-task performance predictions, or holistic player-state modeling.

In its current state, the aim trainer balances short-term satisfaction from completing tasks with long-term measurable progress through global sensitivity graphs and per-task performance histories. This feedback structure differentiates it from typical reaction-based trainers and positions the project as a practical improvement tool. The integration of machine learning further transforms the experience into a personalizable training system for the user to replay over and over to find an optimal sensitivity for all their needs in the first-person shooter genre of games.

Bibliography

Aim Lab. Statespace Labs Inc., 2018. Steam.

Frazier, Peter I. *Lecture 16: Gaussian Processes and Bayesian Optimization*. CS 4787 – Principles of Large-Scale Machine Learning Systems, Spring 2019, Cornell University, 2019, https://www.cs.cornell.edu/courses/cs4787/2019sp/notes/lecture16.pdf. Accessed 22 Nov. 2025.

“Free Gun Sound Effects – Gunshot Sound Effect Downloads – Pixabay.” *Pixabay*, https://pixabay.com/sound-effects/search/gun/.

“Free Mouse Click Sound Effects Download – Pixabay.” *Pixabay*, https://pixabay.com/sound-effects/search/mouse%20click/.

Karray, Yasmine. “Mastering Bayesian Optimization: A Powerful Technique for Efficient Black-Box Function Optimization.” *Medium*, 10 Nov. 2024, https://medium.com/@ykarray29/mastering-bayesian-optimization-a-powerful-technique-for-efficient-black-box-function-optimization-461247e67d76.

“KovaaK’s Scenario Creation: Intro to Scenario Creation.” *KovaaK’s Wiki*, wiki.kovaaks.com/en/home/KovaaK%27s/ScenarioCreation/Intro. Accessed 23 Nov. 2025

Kovaak’s. Version 2.0, The Meta, 2018. Steam.

Maggi, Lorenzo. “A Tutorial on Bayesian Optimization with Gaussian Processes.” *YouTube*, uploaded by Nokia Bell Labs France, https://www.youtube.com/watch?v=VWLI1jthE24.