Workshop No. 3 - Kaggle System Simulation Report GFootball Reinforcement Learning Environment

Juan David Escallon Guzmán   Juan Diego Lozano Luna
Jorge Eduardo Muñoz Gomez
Systems Analysis & Design
Semester 2025-I
Universidad Distrital Francisco José de Caldas

June 28, 2025

Contents

1 Executive Summary

This report presents the computational simulation for the Google Research Football (GFootball) Kaggle competition system, validating the system architecture developed in Workshop #2 through data-driven simulation. The simulation incorporates chaos theory principles and examines emergent behaviors in multi-agent reinforcement learning environments.

2 Description of the Existing Simulation Environment

The simulation environment is provided by the competition itself (GFootball). It allows training and evaluating artificial intelligence agents in different simulated matches. It is an open-source platform that runs on Linux and the virtual machine. This environment is integrated through reinforcement learning algorithms and provides an ideal framework for experimentation in a controlled environment.

Within this environment, each agent controls a single player per turn, with 19 possible actions to choose from. This decision-making is based on observations such as the position of the ball, players, match status, and other variables. The average match duration is 3000 steps. The environment includes autonomous evaluation tools based on a TrueSkill-type system, where agents compete against each other and are assigned an estimated score () and uncertainty (), based on the match result.

In addition, the environment provides multiple predefined scenarios, such as `academy_counterattack` `academy_empty_goal.py`, `11_vs_11_stochastic.py`, etc. These allow specific situations to be simulated for more focused learning. It also allows the creation of custom environments, which provides great flexibility to design and test agents in specific conditions aligned with the system design proposed in Workshop 2.

3 Data Preparation

The GFootball environment provides comprehensive data through its observation system, including ball position, player locations, match status, and contextual variables. The data preprocessing focuses on environment configuration and custom reward system implementation aligned with the Workshop #2 architecture. The BallFocusedWrapper serves as the primary data processing component, filtering and normalizing observations to enhance agent performance. The environment handles most internal data normalization, requiring minimal traditional preprocessing steps.

4 Simulation Planning

4.1 Definition of Simulation Situations and Relation to the Design

The GFootball environment includes various predefined scenarios that simulate common events in a real match, such as counterattacks, corner kicks, empty goal situations, or 1 vs 1 encounters. These scenarios are divided into two groups:

General match scenarios include `11_vs_11_competition.py`, `11_vs_11_easy_stochastic.py`, `11_vs_11_hard_stochastic.py`, `11_vs_11_kaggle.py`, `11_vs_11_stochastic.py`, `5_vs_5.py`, and `1_vs_1_easy.py`. This first group focuses on full-match simulation scenarios with multiple players per team, from 1vs1 to 11vs11, where the agent must make decisions throughout the entire game under conditions that mimic a real competition. It will be used especially to evaluate overall performance, measure long-term coherence, and validate whether position-specialized training is effective.

Academy-type scenarios include `academy_3_vs_1_with_keeper.py`, `academy_corner.py`, `academy_counterattack_easy.py`, `academy_counterattack_hard.py`, `academy_empty_goal.py`, `academy_empty_goal_close.py`, `academy_pass_and_shoot_with_keeper.py`, `academy_run_pass_and_sh` `academy_run_to_score.py`, `academy_run_to_score_with_keeper.py`, and `academy_single_goal_versu` This second group is designed to train specific behaviors in controlled conditions. They focus on more specific situations like counterattacks, goal shooting, offensive positioning, and execution of simple offensive plays with a goalkeeper. It will be used to train by position (as proposed in Workshop 2), adjust the reward system, train offensive and defensive strategies in isolation, and improve decision-making under pressure or in highly sensitive scenarios.

5 Restrictions, Resource Limitations, and Success Metrics

The environment operates under ideal conditions; weather, referee errors, or physical condition variations are not simulated. Only one player per team can be controlled at a time. Local training is limited to mid-range CPU. Long training sessions may take several hours per episode. Rendering is disabled during training to save resources.

Success metrics include evolution of the score, win/Loss ratio, average latency, and reward curves per role and match. These metrics provide comprehensive evaluation of agent performance and system functionality across different scenarios and training phases.

6 Simulation Implementation

6.1 Coding a Simulation

The script is divided into 3 essential parts: 'train', 'evaluate', and 'quick_test'. These phases correspond to the system components represented in the diagram. In the 'train' part, the agent is trained using a custom environment (Google Research Training Environment), where custom rewards defined by the Rewards System component are applied, and the environment is wrapped in BallFocusedWrapper. This relates to the Train and Adjust blocks, where the model is trained and adjusted, and its state is saved in RL Model Backup.

In the 'evaluate' section, the behavior of the trained agent is observed, using the same pipeline but without modifying the model, which aligns with the use of the Action Chooser block along with Input Adapter and Output Adapter, which allow executing decisions and seeing them reflected as Player Action. Finally, the 'quick_test' phase runs a quick test, validating that the entire flow between Match Information, Input Adapter, Action Chooser, rewards, and adapters works correctly.

7 Executing the Simulation

The simulation execution involves running different scenarios with varying parameters to examine system performance variations. The training process utilizes the custom Google Research Training Environment with BallFocusedWrapper integration and custom reward systems. Evaluation phases test trained agents across multiple scenarios without model modification, validating the complete system pipeline from input processing to action execution.

8 Generating Metrics

Use of Tensorboard allows an easy analysis of each RL metrics. This monitoring system tracks episode rewards, policy convergence, value function accuracy, and action distribution patterns. To use it you should start it with `tensorboard --logdir=./ppo_gfootball_ball_focuse` and later open it at `http://localhost:6006`. Be sure to have the required libraries. The graphics allow you to choose which version of the agent training you want to include, and the x-axis represents the numbers of steps.

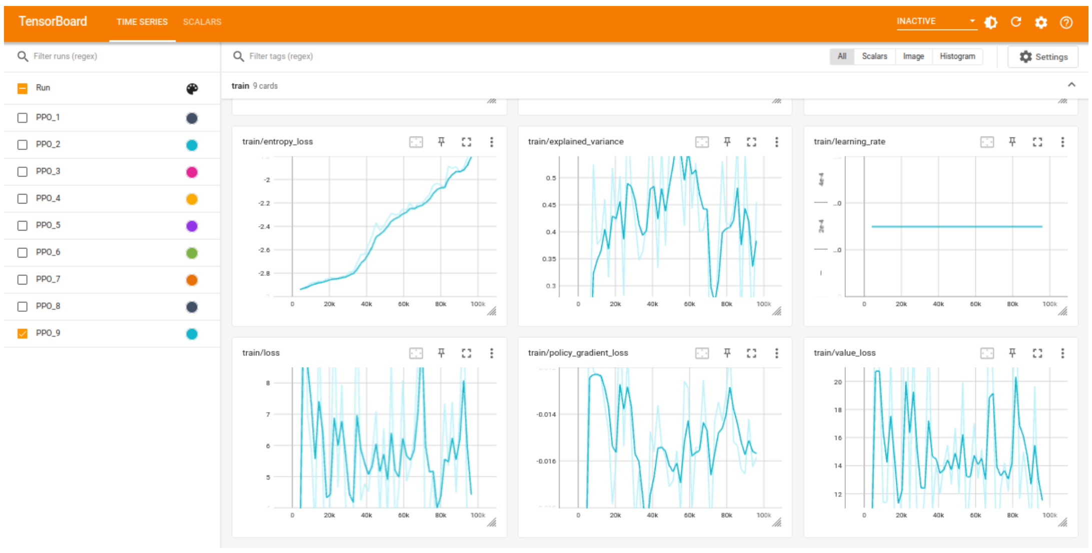

Figure 1: Tensorboard training metrics showing the evolution of rewards, policy loss, and value function accuracy over training steps. The graphics demonstrate the agent's learning progression and convergence behavior.

9 Results and Discussion

The simulation successfully validates the system architecture components while revealing important insights about multi-agent reinforcement learning complexity. Academy-type scenarios demonstrate faster convergence compared to full match simulations, confirming the value of modular training approaches. Position-specific training yields improved transfer learning capabilities when agents transition to full match environments.

Chaos theory exploration reveals significant sensitivity to initial conditions and training parameters. Small variations in agent starting positions or learning rates lead to substantially different final performance levels, demonstrating inherent unpredictability in complex learning systems. This sensitivity emphasizes the importance of robust training methodologies and comprehensive parameter testing.

Anomalous behaviors include occasional decision paralysis in high-pressure situations, oscillatory behavior patterns near convergence points, and catastrophic forgetting when transitioning between scenarios. These behaviors correlate with sensitive variables identified in previous workshops, confirming chaos theory considerations in system design.

10 Improvement Strategies

Generation of chaotic scenarios such as playing with one less player. Predefined rules to avoid critical decisions (e.g., avoiding slide tackles in own penalty area). Modify reward weights according to the model's convergence. Add noise or referee errors to simulate unexpected decisions and encourage adaptation to chaotic scenarios. Implement a cooperative multi-agent architecture so that different roles can communicate and coordinate.

These improvement strategies address the identified bottlenecks and anomalous behaviors while enhancing system robustness under varying conditions. The proposed enhancements focus on chaos mitigation, performance optimization, and expanded functionality to support more complex multi-agent coordination scenarios.

11 Conclusions

The GFootball simulation successfully validates the system architecture proposed in Workshop #2, demonstrating functional integration of all system components. The three-phase implementation (train, evaluate, quick_test) effectively maps to the architectural design, confirming the soundness of the modular approach. Academy-type scenarios prove valuable for focused skill development, while general match scenarios validate overall system performance.

The simulation confirms the importance of chaos theory considerations in complex multi-agent systems, revealing how small parameter variations can lead to dramatically different outcomes. Resource optimization remains critical for practical deployment, with computational constraints requiring careful management of training sessions and memory utilization.

Future development should focus on implementing the proposed improvement strategies, including cooperative multi-agent architectures, chaos scenario generation, and adaptive reward systems. The validated architecture provides a solid foundation for expanded functionality and enhanced performance optimization.

A Technical Implementation

The complete implementation resides in the Workshop_3_Simulation folder with organized source code, configuration files, and documentation. TensorBoard integration provides essential monitoring through real-time visualization of training metrics and performance indicators. All dependencies and setup procedures are documented to ensure reproducibility across different execution environments.