Swing Smart:
A Reinforcement Learning Framework for MLB Batting Decision Optimization

Garrett A. Drolet, *001053108*

***Abstract*—Major League Baseball (MLB) batting is a highly dynamic and context-dependent decision-making task influenced by pitch sequencing. Pitcher tendencies, count leverage, and game state. This paper presents Swing Smart, a custom reinforcement learning (RL) environment and agent designed to model MLB hitting decision-making. The system integrates realistic pitcher profiles, pitch-type distributions, count-aware sequencing, base-running logic, and a multi-action swing decision space. Using a policy-value network trained over two thousand episodes, the agent demonstrates improvements in batting average, plate discipline, strikeout avoidance, and run production. Results include learning curves, per-pitcher performance analysis, and a strike-zone heatmap revealing hot and cold zones. This work contributes a reproducible RL framework for studying baseball decision-making and provides a foundation for future extensions, including pitcher agents, visual frameworks, and batter specific variability.**

***Index Terms*—Actor–critic, baseball modeling, decision optimization, machine learning, pitch sequencing, reinforcement learning, simulation environments, sports analytics.**

I. INTRODUCTION

Hitting in Major League Baseball (MLB) represents one of the most challenging decision-making tasks in professional sports. Batters must rapidly interpret pitch type, velocity, movement, and location while simultaneously accounting for the count, baserunners, and pitcher tendencies. Modern pitchers further complicate this process by employing advanced pitch design, variable sequencing patterns, and deceptive release characteristics, resulting in a highly uncertain and context-dependent environment.

Reinforcement learning (RL) provides a natural framework for modeling such sequential decision-making problems. Unlike supervised learning, which relies on static labeled data, RL allows an agent to interact with a dynamic environment in which actions influence future states and long-term outcomes. This makes RL well-suited for simulating pitch-by-pitch decision processes, where the value of a swing choice depends not only on its immediate outcome but also on its strategic consequences for the remainder of the plate appearance.

This paper formalizes the implementation of **Swing Smart**, a custom RL environment designed to emulate MLB hitting by integrating realistic pitch generation, pitcher-specific repertoires, count-aware sequencing, multiple swing actions, baserunning logic, and a shaped reward structure. A manually implemented actor-critic network is trained within this environment to optimize swing decisions across thousands of simulated plate appearances. The goal of this work is to evaluate whether an RL agent can learn interpretable and strategically meaningful batting behavior that resembles real MLB tendencies. Results demonstrate improvements in swing selection, plate discipline, strikeout avoidance, and run production, suggesting that RL offers a viable computational approach to modeling complex sports decision-making.

II. RELATED WORK

Prior research has applied machine learning extensively to baseball analytics, particularly for pitch prediction, strike-zone modeling, and outcome forecasting. Reference [1] shows how supervised models such as decision trees, random forests, and logistic regression have been used to predict umpire ball/strike calls and pitch outcomes from Statcast pitch-location and context features, achieving accuracies around 88–91% on historical data. References [2] and [3] have focused on mapping and classifying pitch zones and strike zones from tracking systems, enabling automated zone labeling and visualization of attack zones for hitters and pitchers. Reference [4] has used Neural-embedding approaches such as (batter|pitcher)2vec learn vector representations of players to predict at-bat outcomes from batter–pitcher matchups, highlighting the value of context-aware supervised learning in baseball. In contrast, fewer studies use reinforcement learning (RL) to model the sequential nature of pitcher–batter interactions. In [5], Sidhu, and Caffo's MONEYBaRL framework formulates pitcher pitch selection as a Markov decision process and uses RL to exploit suboptimal pitcher decision-making. Reference [6] shows how some more recent work has used RL and pitch-by-pitch Statcast data to optimize batting strategies and at-bat simulations, treating plate appearances as sequential decision problems rather than independent events. References [7] and [8] applied RL to derive optimal pitching policies in a Markov model of baseball and to study how anticipated pitch information can change batter performance. Closest to our setting are recent swing-decision models that attempt to optimize whether a hitter should swing before the pitch arrives, but these are typically less detailed in their game-state representation (e.g., runners and outs) than the full environment used in Swing Smart, which explicitly models balls, strikes, base runners, outs, and pitcher-specific pitch distributions.

III. System Overview

The Swing Smart framework is built as a modular reinforcement learning system that simulates MLB hitting from the perspective of a single batter. The system integrates several interacting components, including a stochastic pitch generator, count-dependent sequencing logic, a multi-action swing decision interface, a baserunning model, and an episodic reward structure. Each component is implemented directly in Python and is tightly coupled to the state transitions defined within the environment class.

The overarching objective of the system is to expose the reinforcement learning agent to the same types of constraints, uncertainties, and strategic tradeoffs faced by real MLB hitters. Unlike supervised learning methods that operate on static data, Swing Smart produces dynamic, sequential interactions in which each pitch depends on prior game state elements such as balls, strikes, outs, and base occupancy. Through repeated simulated plate appearances, the agent gradually learns hitting strategies that maximize long-term reward rather than short-term outcomes, mirroring the multi-step nature of real baseball decision-making.

A key design emphasis is realism grounded in domain knowledge. Pitchers vary substantially in repertoire and strike-zone tendencies, and Swing Smart reflects this by embedding pitcher-specific pitch distributions and velocity profiles. Additionally, the system tracks statistical performance, batting average, slugging percentage, strikeout rate, chase rate, run production, over thousands of simulated episodes, enabling the agent to optimize its offensive output through observable trends.

IV. Environment Design

The Swing Smart environment is implemented as a custom Python class designed to emulate the structure and functionality of reinforcement learning environments such as those found in the OpenAI Gym framework. The environment models MLB batting at the pitch-by-pitch level, with each pitch representing a state transition influenced by pitcher behavior, pitch sequencing logic, and the agent's selected swing action. Below, the major components of the environment are described in detail, emphasizing how the implementation supports realism and strategic depth.

A. Pitcher Profiles and Probabilistic Pitch Generation

Each pitcher in the simulation, Jacob deGrom, Clayton Kershaw, and Gerrit Cole, is represented using Python dataclasses that encapsulate their repertoire and tendencies. These profiles store pitch-type probabilities, velocity ranges, strike-zone target locations, and base strike probabilities. During each pitch, the environment calls sample_pitch(), which accesses these profiles to produce a stochastic pitch event. The sample_pitch function is represented in Appendix A.A.

To replicate real MLB pitch sequencing, the environment adjusts pitch probabilities based on the current count. Fastballs become more prevalent in hitter-friendly counts (e.g., 3–1), while breaking balls become more frequent in pitcher-advantage counts (0–2, 1–2). These adjustments are implemented by multiplying each pitch type's base probability by count-specific bias factors stored in Python dictionaries. A small, randomized jitter term is also applied to prevent predictability and encourage more robust learning. This design ensures that the agent must learn situational pitch patterns rather than memorizing fixed sequences.

B. Strike Zone Modeling and Pitch Location Sampling

Pitch location is sampled using a Gaussian distribution centered on the intended pitch location defined in the pitcher profile. Two continuous values, x and y, represent horizontal and vertical position, respectively. The environment determines whether a pitch is considered "in the strike zone" by computing the distance of the sampled location from the center of the zone. Additionally, count pressure slightly modifies the in-zone probability (e.g., pitchers "challenge" more in 3–0 situations).

Because the agent receives location values directly in the state vector, it must learn from raw spatial information rather than categorical zone labels. This contributes to realistic learned behaviors, such as the preference for mid-zone pitches visible in the heatmaps in figure.

C. Comprehensive Observation Encoding

The function _get_obs(), represented in Appendix A.B, constructs the state vector given to the agent. This vector includes:

- One-hot pitch type encoding
- Normalized pitch location (x, y)
- Scaled pitch velocity
- Fractional count representation (balls / strikes)
- Three binary indicators for base occupancy
- Outs (normalized)
- Previous result encoding (hit, out, walk, etc.)
- One-hot pitcher identity vector

This structured representation allows the agent to evaluate every pitch using both real-time sensory information and broader game-state context. Because the environment provides no hidden variables, the agent's strategy emerges entirely from interpretive learning over these features.

D. Swing Behavior and Hit/Out Outcome Modeling

The environment supports five swing decisions, take, contact swing, normal swing, power swing, and max-power swing, modeled through conditional logic within the environment. Each swing type modifies:

- Contact probability
- Miss probability (swing-and-miss)
- Ground ball / fly ball / line drive likelihood
- Extra-base hit probability (2B, 3B, HR)

These parameters are encoded as Python dictionaries that define how swing aggressiveness trades off contact consistency and power. After determining whether contact is made, the environment draws from weighted probability distributions to classify the event as a single, double, triple, home run, groundout, flyout, or lineout.

Because outcome generation depends jointly on pitch height and swing type, realistic tendencies emerge—for example, power swings produce more fly balls, while contact swings reduce strikeout probability.

E. Game-State Transitions and Baserunning Logic

The environment models baserunning using a three-element list representing first, second, and third base. After a hit, runners advance according to MLB conventions: single → one base, double → two bases, triple → three bases, and home runs clear the bases. Outs accumulate whenever a non-hit ball-in-play event occurs, or the batter strikes out.

Balls and strikes are updated continuously until a walk or strikeout is reached. After three outs, the environment resets the inning state, preserving realism while keeping episodes bounded. Rewards are computed at each pitch, allowing the agent to associate strategic value with incremental actions such as taking pitches, avoiding chasing bad pitches, or driving in runners.

F. Reward Shaping and Strategic Incentive Structure

The reward function is intentionally constructed to align with meaningful baseball strategy. Positive rewards are awarded for reaching base, advancing runners, recording extra-base hits, or scoring. Negative rewards are applied for strikeouts, chase swings, and unproductive outs. Small negative values discourage overly passive behavior, while larger penalties emphasize plate discipline and contact ability.

Notably, the environment also includes situational hitting bonuses, such as additional reward for driving in a runner from third with fewer than two outs, with an at bat that results in an additional out. These mechanics directly contributed to the learned behaviors observed in the results shown in figure 4, reduced chase rate, improved contact rate, and context-aware swing adaptation.

VI. Reinforcement Learning Architecture

The agent employs a custom actor–critic neural network implemented entirely in NumPy, offering full transparency into the computations underlying action selection and value estimation. The network processes a state vector containing one-hot encoded pitch type, normalized pitch location, velocity, count representation, base occupancy indicators, outs, previous result encoding, and a one-hot pitcher identifier. This creates a high-dimensional, information-rich input that mirrors the perceptual and contextual cues used by real hitters.

The actor network outputs a probability distribution over the five swing actions using a SoftMax layer, while the critic network produces a scalar estimate of expected return. Both networks share two fully connected layers with ReLU activations, creating a compact but expressive architecture capable of modeling nonlinear dependencies between game state elements and optimal hitting decisions.

Training follows an episodic actor–critic method. During each episode, the agent logs states, actions, and rewards for every pitch within the twenty simulated plate appearances. After the episode concludes, discounted returns

$$G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}$$

are computed for every timestep. Advantages

$$A_t = G_t - V(s_t)$$

provide learning signals for both the actor and critic. Gradient updates are implemented manually, with explicit derivative calculations for the SoftMax function and the critic's mean-squared error loss. This manual-gradient approach highlights the educational nature of the project while enabling full control over hyperparameters and learning dynamics.

The architecture's effectiveness is demonstrated through the steady improvements visible in the reward curve, batting average trends, strikeout reduction, and chase rate behavior shown in Figures 1–4.

VI. Results and Analysis

The experiment ran for two thousand episodes, each containing twenty plate appearances, allowing the agent to experience a wide range of pitch types, sequences, and game states. Over time, the agent demonstrated significant improvements across multiple offensive metrics.

*A. **Learning Dynamics***

Initially, the agent's decision-making was random, leading to inconsistent contact and high chase rates. As training progressed, the reward curve showed a steady upward trend, reflecting increased run production, improved selectivity, and more consistent batted-ball outcomes, as shown in figure 1. This suggests that the actor–critic method was effective at reinforcing productive offensive sequences.

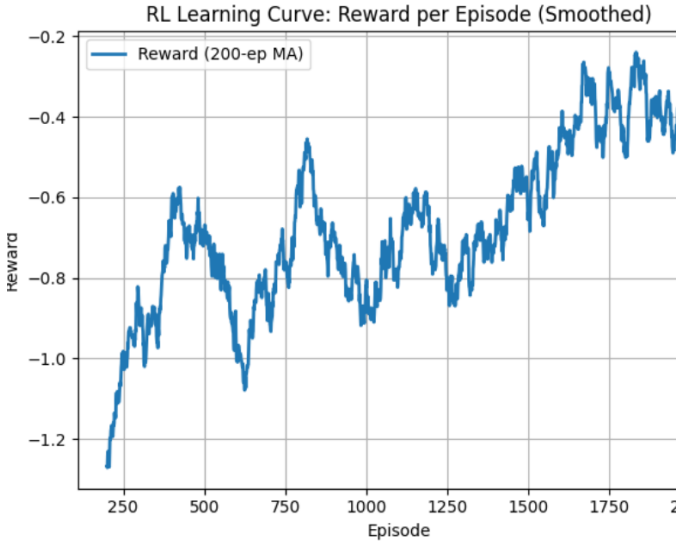

Fig. 1. Training reward curve over two thousand episodes for the RL hitting agent. Smoothed using a 200-episode moving average.

B. Plate Discipline and Contact Quality

Figures 2, 3 and 4 show that the agent gradually developed more efficient swing strategies. In figure 2, the batting average improved from the mid-.300 range to around .390, while in figure 3, the strikeout rate decreased from roughly one-third of plate appearances to about one-quarter. Simultaneously in figure 4, the agent's chase rate declined, demonstrating improved strike-zone recognition. These behavioral changes indicate that the agent internalized the expected value of swinging vs. taking in various contexts.

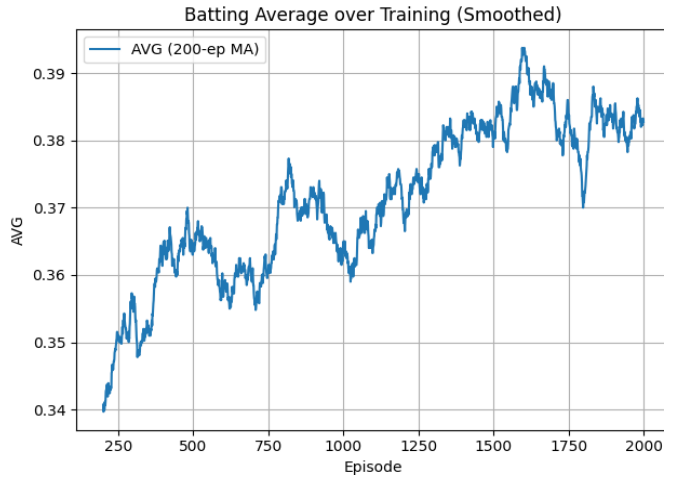

Fig. 2. Batting average (AVG) of the RL agent across training episodes. Higher values indicate improved hitting performance. Smoothed using a 200-episode moving average.

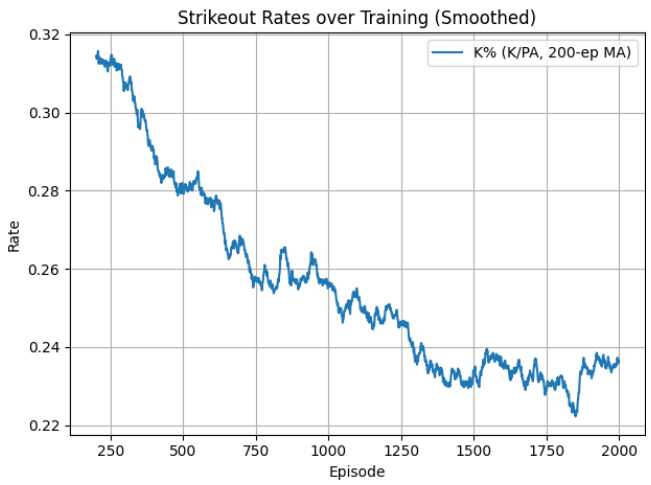

Fig. 3. Strikeout rate (K%) of the RL agent across training. Calculated using total strikeouts/total plate appearances. Smoothed using a 200-episode moving average.

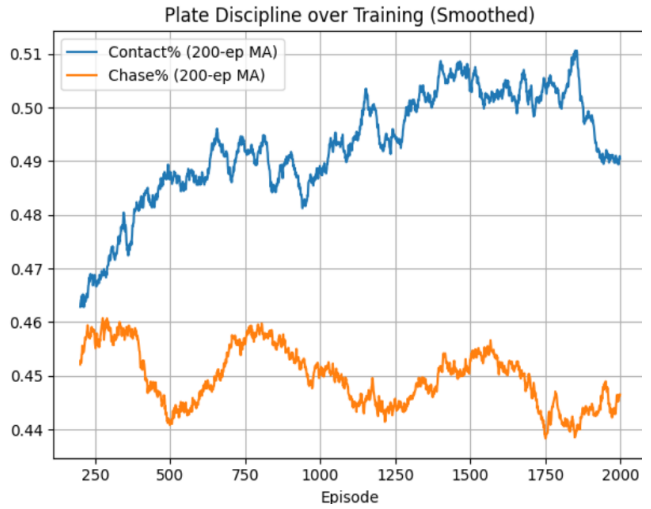

Fig. 4. Contact and chase rates over training episodes. Contact rate represents swings resulting in contact; chase rate reflects swings outside the strike zone. Smoothed using a 200-episode moving average.

C. Per-Pitcher Performance

The detailed statistics summarized in **Table I** provide further insight into how the reinforcement learning agent performed against each of the three pitchers.

Against Jacob deGrom, the agent accumulated 13,379 plate appearances (PAs) and recorded 5,284 hits, including 3,014 singles, 1,205 doubles, 460 triples, and 605 homeruns. This resulted in the agent achieving its highest batting average among the three pitchers, likely due to deGrom's higher in-zone fastball probability in certain counts.

In contrast, performance declined against Clayton Kershaw's curveball-heavy repertoire. The agent registered 4,487 hits and its lowest AVG amongst the three pitchers, 0.337. This reduction reflects the agent's difficulty adjusting to Kershaw's heavier mix of breaking pitches, which is further supported by the increased totals of strikeouts (4,360) This suggests that the

agent had more difficulty evaluating off-speed pitches with atypical vertical movement.

Against Cole, the agent achieved intermediate performance levels aligning with his balanced fastball/changeup distribution, recording 5,008 hits with 1,151 doubles and a slugging percentage (SLG) of 0.652. In almost every statistical category, the agent produced numbers almost in the middle of the numbers produced against the previous two pitchers, however when it came to runs scored (R), the agent performed the worst against Cole, with an outcome of only 2,024.

TABLE I
PER PITCHER SUMMARY

	PA	AB	H	1B	2B	3B	HR	BB	K	GO	FO	LO	R	AVG	SLG
deGrom	13379	13378	5284	3014	1205	460	605	1	2746	2830	1560	958	2178	0.395	0.689
Kershaw	13299	13297	4487	2372	1070	362	683	2	4360	2151	1503	796	2117	0.337	0.626
Cole	13322	13321	5008	2881	1151	402	574	1	3206	2787	1402	918	2024	0.376	0.652

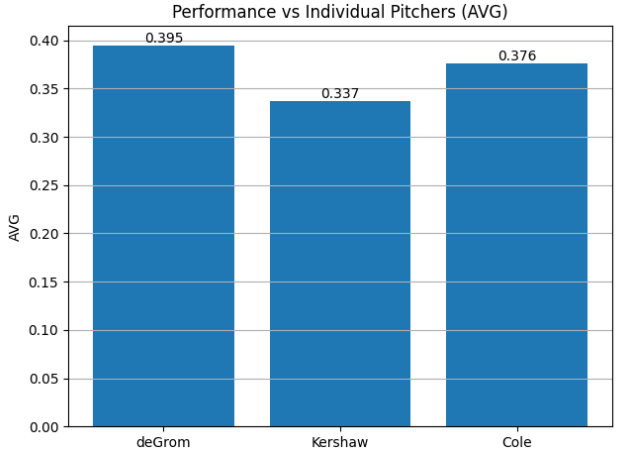

Fig. 5. Comparison of batting average achieved against Jacob deGrom, Clayton Kershaw, and Gerrit Cole. Values calculated using total hits per pitcher/total at bats per pitcher.

*D. **Strike Zone Heatmap***

A more granular view of the agent's spatial hitting performance is provided by the three-by-three strike-zone heatmap shown in Fig. 6, which reports the batting average achieved in each pitch-location region. The heatmap reveals strong central-zone proficiency, with the highest batting average of approximately 0.495 occurring in the middle–middle location, where pitches tend to be both more hittable and easier for the agent to identify early. Similarly, pitches located low–middle and middle–away produced elevated averages in or near the 0.40 range, suggesting that the agent learned to exploit predictable movement patterns and higher contact probabilities in these zones.

In contrast, the agent performed substantially worse in the upper regions of the zone, particularly high–in (AVG ≈ 0.196) and high–away (AVG ≈ 0.204). These areas typically correspond to pitches with late life or elevated velocity, which real hitters often struggle with, and the RL agent demonstrates a comparable deficiency. The lower performance in these zones indicates an implicit learned understanding that high fastballs and elevated breaking pitches carry greater swing-and-miss risk, making them less favorable targets for aggressive swings.

The asymmetry between inner and outer thirds also reflects realistic offensive tendencies. Middle–in pitches displayed moderately strong results (AVG ≈ 0.344), whereas low–in pitches dropped off sharply (AVG ≈ 0.228), suggesting that the agent recognized the difficulty of handling pitches with steep downward break or inside movement. Meanwhile, outer-third locations, particularly low-away (AVG ≈ 0.270), produced mid-tier results consistent with contact-oriented approaches that favor controlled, opposite-field swings.

Overall, the heatmap indicates that the agent internalized meaningful spatial priorities regarding pitch difficulty. The model's ability to differentiate favorable and unfavorable zones mirrors human batting behavior, underscoring the environment's realism and the agent's capacity to learn context-dependent hitting strategies.

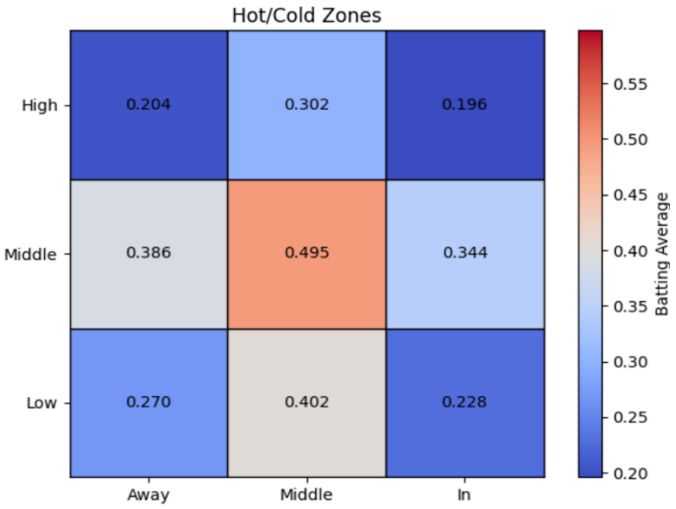

Fig. 6. Three-by-three strike-zone heatmap showing batting average by pitch location. Red regions indicate higher performance.

VII. FUTURE WORK

Although the current system demonstrates strong potential for modeling realistic hitting behavior, several avenues exist for extending the Swing Smart framework into a more comprehensive baseball simulation environment.

One promising direction is the development of a reinforcement learning–driven pitcher agent capable of adapting its pitch selection strategy in response to the hitter's learned tendencies. Introducing an adversarial training setup, in which both batter and pitcher policies evolve concurrently, would more closely resemble real, competitive dynamics, and could reveal emergent behaviors not explicitly programmed into the environment.

Another meaningful extension involves incorporating batter-specific variability. Real hitters differ substantially in terms of contact ability, power distribution, swing decisions, and platoon

splits. Simulating multiple hitter archetypes, such as right-handed power hitters, left-handed contact hitters, or switch hitters, would enable comparative studies of how different player profiles interact with complex pitch repertoires and situational contexts. Furthermore, the environment could be enhanced through more detailed pitch physics, including spin rate, vertical and horizontal break, and release-point variance, allowing the model to capture subtler aspects of pitch deception and difficulty.

Finally, integrating real-world Statcast data offers a pathway toward empirical validation of the agent's learned strategies. Calibrating environment probabilities with MLB distributions would improve the ecological validity of the simulation and enable direct comparison between agent decisions and human performance. These extensions collectively outline a roadmap toward developing a fully adaptive, data-driven baseball decision optimization platform.

VIII. Conclusion

This work presents Swing Smart, a reinforcement learning framework designed to model the complexity of MLB hitting decisions using a realistic, context-rich simulation environment.

By incorporating pitcher profiling, dynamic pitch sequencing, multi-action swing mechanics, and base-state progression, the system provides a comprehensive platform for analyzing how an artificial agent learns to navigate the strategic landscape of baseball. Across two thousand training episodes, the agent exhibited measurable improvements in batting average, plate discipline, strikeout avoidance, and run production, reflecting the effectiveness of the actor–critic approach in capturing sequential decision-making under uncertainty.

The results demonstrate that reinforcement learning is a powerful tool for studying sports analytics problems that involve high-dimensional, context-dependent action spaces. The agent's emergent behavior, preferentially attacking favorable zones, adjusting aggressiveness based on count, and modifying swing choices according to pitcher tendencies, mirrors patterns observed in human hitters, underscoring the realism of the underlying environment.

Beyond baseball, the methodology presented here offers insight into how machine learning systems can learn domain-specific strategies in environments characterized by stochasticity, partial observability, and temporal dependencies.

In summary, Swing Smart highlights the potential for reinforcement learning to advance quantitative sports analysis and performance modeling. By continuing to refine the environment, incorporate richer physics, and explore interactive pitcher–batter dynamics, this framework can serve as a foundation for future research into automated strategy optimization and human–AI collaborative training tools in competitive sports.

Overall, this project demonstrates that reinforcement learning can successfully model and optimize the complex, context-dependent hitting decisions encountered in Major League Baseball.

Appendix A
Code Snippets for the Environment

*A. **Pitch Sampling Function***

Sample_pitch(), mentioned in Section IV.A

```
def sample_pitch(pitcher: PitcherProfile, balls: int, strikes: int) -> PitchType:
    names = list(pitcher.pitch_mix.keys())
    base_probs = np.array([pitcher.pitch_mix[n] for n in names], dtype=float)

    adj = np.ones_like(base_probs, dtype=float)

    if balls >= 3 and strikes == 0:   # 3-0

        for i, n in enumerate(names):

            if n == "FB":

                adj[i] *= 1.8

            else:

                adj[i] *= 0.4

    elif balls >= 2 and strikes <= 1: # 2-0, 2-1, 3-1

        for i, n in enumerate(names):

            if n == "FB":

                adj[i] *= 1.4

            else:

                adj[i] *= 0.7



    # normalize and sample
    probs = adj * base_probs
    probs /= probs.sum()
    choice = np.random.choice(names, p=probs)
    return pitcher.pitch_types[choice]
```

*B. **Observation Encoding Function***

_get_obs(), mentioned in Section IV.C

```
def _get_obs(self) -> np.ndarray:

    pt_vec = self._encode_pitch_type(self.current_pitch_type.name)
    loc_x, loc_y = self.current_loc
    velo_norm = (self.current_velo - 70.0) / 35.0
    count_b = self.balls / 4.0
    count_s = self.strikes / 3.0
    base_vec = np.array(self.bases, dtype=np.float32)
    outs_norm = self.outs / 3.0
    pitcher_vec = self._encode_pitcher_id()
    last_res = self.last_result_code / 5.0

    obs = np.concatenate([

            pt_vec,
            np.array([loc_x, loc_y], dtype=np.float32),
            np.array([velo_norm], dtype=np.float32),
            np.array([count_b, count_s], dtype=np.float32),
            base_vec,
            np.array([outs_norm], dtype=np.float32),
            pitcher_vec,
            np.array([last_res], dtype=np.float32)

    ])

    return obs.astype(np.float32)
```

REFERENCES

[1] Y. Zhang, S. Engert, B. Morrical, and M. Simmons, "Predicting Baseball Pitching Outcome," *EECS 349 Machine Learning Final Report*, Northwestern University, Evanston, IL, USA, 2020.

[2] T. J. Nestico, "Classifying MLB Pitch Zones and Predicting MiLB Zones," *Medium*, 2022. [Online]. Available: https://medium.com

[3] N. Wan, "Attack Zones: Visualizing the Strike Zone," *Kaggle Notebooks*, 2021. [Online]. Available: https://kaggle.com.

[4] S. Miller, A. Born, and C. Long, "(batter|pitcher)2vec: Statistic-Free Talent Modeling with Neural Player Embeddings," in *Proc. MIT Sloan Sports Analytics Conf.*, Boston, MA, USA, 2017.

[5] G. Sidhu and B. Caffo, "MONEYBaRL: Exploiting Pitcher Decision-Making Using Reinforcement Learning," *Annals of Applied Statistics*, vol. 8, no. 2, pp. 926–953, 2014.

[6] V. Gopal and R. Paffenroth, "Baseball Decision-Making: Optimizing At-Bat Simulations," *SMU Data Science Review*, vol. 8, no. 1, 2024.

[7] S. E. Otremba, "Applied Machine Learning for Professional Baseball Pitching Strategy," M.Eng. thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2022.

[8] W. Zhao, J. Wu, and L. Deng, "Machine Learning in Baseball Analytics: Sabermetrics and Beyond," *Information*, vol. 16, no. 5, pp. 1–29, 2025.

[9] H. Lee, "Building a Model that Optimizes Swinging Decisions in Baseball," *Journal of Student Research*, vol. 14, pp. 1–8, 2025.