Adaptive Broad-Phase Collision Detection in Unity Using Thompson Sampling

Raul Costa Feitosa[1], **Maria Andréia Formico Rodrigues**[1]

[1]PPGIA & GIRA Lab
Universidade de Fortaleza (UNIFOR)
Av. Washington Soares, 1321 – 60811-905 – Fortaleza – CE – Brazil

raulcostaf@edu.unifor.br, mafr@unifor.br

Abstract. Introduction: *Broad-phase collision detection strongly affects frame rate, memory use, and responsiveness in games and interactive simulations. However, the best strategy depends on runtime conditions such as object count, spatial distribution, and motion.* ***Objective:*** *To present an adaptive mechanism that selects broad-phase collision detection algorithms at runtime using Thompson Sampling.* ***Methodology:*** *The system was implemented as a Unity plugin linked to a native broad-phase library containing Brute Force, Grid, SAP, AxisSweep, DBVT, Tracy, and KDTree. Selection uses a multi-metric reward based on FPS, detection time, memory use, and Axis-Aligned Bounding Box (AABB) checks, together with adaptive FPS normalization, cooldown control, and post-switch stabilization. Evaluation covered Brownian Motion, Rotating Gravity, and Hurricane scenarios with 1,000, 2,000, and 3,000 heterogeneous dynamic objects.* ***Results:*** *Across five runs per condition, AxisSweep dominated Brownian Motion with 1,000–2,000 objects and Hurricane with up to 2,000 objects. KDTree dominated Hurricane and Rotating Gravity with 3,000 objects, while Brownian Motion with 3,000 objects and Rotating Gravity with 1,000 objects showed mixed dominance. Results suggest that Thompson Sampling is a promising strategy for adaptive broad-phase selection, especially in workloads where the best spatial data structure changes.*
Keywords *Broad-Phase Collision Detection, Spatial Data Structures, Thompson Sampling, Multi-Armed Bandits, Game Physics, Runtime Adaptation.*

1. Introduction

Collision detection is a core component of physics engines, game engines, and interactive simulations [Ericson 2004]. Its broad-phase stage quickly discards object pairs that are far apart, reducing the number of expensive narrow-phase tests and directly affecting frame rate, memory use, and responsiveness. However, no single broad-phase strategy is best for all workloads. Brute-force checks, grids, sweep-based methods, dynamic bounding volume trees, and KD-trees differ in insertion cost, query cost, memory overhead, and sensitivity to object count, spatial distribution, and motion pattern [Coumans 2018].

Most game engines use a fixed broad-phase configuration selected at design time or manually by the developer. This static choice can become suboptimal when the runtime workload differs from the expected scenario [Rice 1976]. Hand-crafted switching rules are possible, but they are difficult to tune and may not generalize across scenes with different densities, movement patterns, or load spikes. This motivates treating broad-phase selection as an online decision problem.

Multi-Armed Bandits (MABs) provide a compact formulation for online selection under partial feedback [Lattimore and Szepesvári 2020]. At each decision cycle, the system observes only the performance of the active algorithm, while the rewards of inactive algorithms remain unknown. Thompson Sampling is a Bayesian MAB strategy that balances exploration and exploitation through posterior sampling, without requiring manually defined exploration schedules [Thompson 1933, Chapelle and Li 2011, Russo et al. 2018].

This paper presents an adaptive mechanism for runtime selection of broad-phase collision detection algorithms in a Unity-based simulation plugin [Feitosa 2026]. The work focuses on the selection mechanism, not on proposing a new broad-phase data structure. The main contributions are: (i) a Thompson-Sampling-based decision model using a multi-metric reward; (ii) robustness mechanisms for adaptive FPS normalization, cooldown, post-switch stabilization, and real-time bail-out; (iii) a Unity plugin architecture connecting a real-time simulation to a native broad-phase algorithm library; and (iv) an experimental evaluation showing how posterior confidence can support both algorithm selection and workload-ambiguity diagnosis under changing scene dynamics.

2. Background & Related Work

Broad-phase collision detection reduces the number of object pairs passed to the narrow phase and is essential to real-time games and simulations [Ericson 2004]. Common approaches include exhaustive testing, uniform grids, SAP and AxisSweep, DBVTs, KD-trees, and hybrid methods such as Tracy [Lo et al. 2013, Tracy et al. 2009, Kopta et al. 2012, Serpa and Rodrigues 2019]. Their performance depends on factors such as object count, spatial distribution, clustering, motion coherence, insertion and removal patterns, and update costs. Therefore, the most suitable algorithm may vary across workloads.

The algorithm selection problem states that no single algorithm is best for all inputs [Rice 1976]. At runtime, this becomes an online decision problem in which the system selects an algorithm, observes its performance, and updates future choices. A MAB is suitable because only the selected algorithm is evaluated at each cycle [Bubeck and Cesa-Bianchi 2012, Lattimore and Szepesvári 2020]. This avoids the additional state modelling and training complexity of full reinforcement learning while preserving exploration and exploitation.

Previous studies have evaluated broad-phase algorithms using synthetic workloads, physics simulations, GPU experiments, and benchmarking frameworks [Woulfe and Manzke 2009, Liu et al. 2010, Lo et al. 2013, Serpa and Rodrigues 2020]. Engines and tools such as Bullet, PhysX, and PEEL also provide comparative environments [Coumans 2018, NVIDIA 2019, Terdiman 2017, Serpa and Rodrigues 2019]. However, these works mainly support offline comparison or fixed-algorithm evaluation. Adaptive broad-phase methods react to changing scene conditions [Avril et al. 2014, Capannini and Larsson 2017], but typically adjust the internal configuration of a single method. In contrast, our work considers complete broad-phase algorithms as runtime alternatives. Online algorithm selection addresses repeated decisions under changing conditions and partial feedback [Tornede et al. 2022], matching

a runtime setting in which only the active algorithm is observed. Possible online selection policies include Upper Confidence Bound (UCB), epsilon-greedy, contextual bandits, and heuristic switching [Bubeck and Cesa-Bianchi 2012, Lattimore and Szepesvári 2020]. Unity and native-library integration further enable engine-level runtime extensions.

Prior work has not evaluated online learning-based selection among structurally distinct broad-phase algorithms within a running game engine. This setting involves partial feedback, switching overhead, warm-up effects, and changing workloads. We therefore investigate Thompson Sampling as an online selector among seven broad-phase algorithms integrated into Unity, focusing on their runtime integration and empirical analysis rather than on Thompson Sampling itself.

3. Candidate Broad-Phase Algorithms

The native library [Serpa and Rodrigues 2019] used here offers 7 candidate broad-phase strategies: BruteForce, Grid, Sweep and Prune (SAP), AxisSweep, Dynamic Bounding Volume Tree (DBVT), Tracy, and KDTree. Together, they cover exhaustive, spatial subdivision, sweep-based, hybrid, and hierarchical approaches, exposing different trade-offs between update cost, query cost, memory use, and sensitivity to spatial distribution. Table 1 summarizes their evaluation roles.

Table 1. Candidate broad-phase algorithms considered by the selector.

Algorithm	Main idea	Expected behavior
BruteForce	Tests all object pairs.	Simple baseline; poor scalability.
Grid	Fixed spatial subdivision.	Efficient for uniform distributions; weak under clustering.
SAP/AxisSweep	Ordered interval sweeping.	Benefits from temporal coherence and smooth motion.
DBVT	Dynamic bounding hierarchy.	Handles dynamic scenes but requires hierarchy maintenance.
Tracy	Spatial subdivision plus local sweep.	Combines locality with temporal coherence.
KDTree	Hierarchical spatial partitioning.	Useful for non-uniform or locally clustered workloads.

4. Adaptive Broad-Phase Selection Mechanism

This mechanism formulates runtime broad-phase selection as a MAB problem. Each candidate algorithm is modeled as an arm:

$$\mathcal{A} = \{\mathrm{BruteForce}, \mathrm{Grid}, \mathrm{SAP}, \mathrm{AxisSweep}, \mathrm{DBVT}, \mathrm{Tracy}, \mathrm{KDTree}\}.$$

At each decision instant, only the active algorithm is observed. For algorithm a, the system collects $\mathbf{m}_a(t_k) = (\mathrm{FPS}_a, \mathrm{DET}_a, \mathrm{MEM}_a, \mathrm{CHK}_a)$, where FPS is frame rate, DET is detection time, MEM is memory overhead, and CHK is the number of Axis-Aligned Bounding Box (AABB) checks. These metrics are normalized to $[0, 1]$ and combined into the scalar reward

$$r_a(t_k) = w_{FPS}\hat{f}_{FPS} + w_{DET}\hat{f}_{DET} + w_{MEM}\hat{f}_{MEM} + w_{CHK}\hat{f}_{CHK}.$$

We use $w_{FPS} = 0.55$, $w_{DET} = 0.25$, $w_{MEM} = 0.12$, and $w_{CHK} = 0.08$, prioritizing frame rate while still accounting for detection latency, memory, and pruning efficiency. These user-configurable weights can be adjusted through the plugin script to reflect application-level priorities rather than universal constants, with FPS emphasized to preserve interactive responsiveness. FPS is normalized adaptively: instead of using a fixed absolute target, the score is computed relative to the recent performance range observed in the current run, while preserving a lower bound for near-stall cases. This keeps the reward informative in both light and dense scenes. For each arm, the selector maintains a Beta posterior $\mathrm{Beta}(\alpha_a, \beta_a)$. All arms start with $\alpha_a = \beta_a = 1$ unless historical priors are available. After observing reward r_a, only the active arm is updated: $\alpha_a \leftarrow \alpha_a + r_a(t_k), \qquad \beta_a \leftarrow \beta_a + (1 - r_a(t_k))$. This posterior representation also makes the selector interpretable: high posterior means with large accumulated evidence indicate stable preference, whereas low means or limited evidence indicate uncertainty or workload ambiguity. Thompson Sampling then draws one sample from each posterior and selects the arm with the highest sampled value as the next candidate. A switch is executed only when a cooldown interval has elapsed. This formulation approximates a non-stationary runtime setting, since observed rewards may depend on warm-up, temporal coherence, and switching costs. The current non-contextual model does not explicitly represent scene or backend state.

Two safeguards improve stability. First, the selector collects metrics continuously but evaluates candidates only at fixed intervals, with cooldown control limiting switches. After each switch, it resets metric histories and counters and pauses Bayesian updates during a minimum settling window, preventing transient frames from biasing posterior estimates and reducing oscillations. Second, if a frame exceeds five seconds, a real-time bail-out penalizes the active arm with ten zero-reward observations and allows immediate resampling without waiting for the cooldown. Together, these safeguards preserve adaptivity while limiting transition bias and pathological behavior.

Broad-phase algorithms maintain distinct internal structures, such as sorted lists, trees, and grids, and therefore incur transition overhead when switched. The current prototype does not migrate state between backends. Instead, the selected algorithm receives the current Unity-side AABB buffer and initializes or updates its native structure. A rebuild may be required upon first activation or when the object count changes, temporarily reducing temporal coherence and introducing warm-up costs.

5. The Proposed Plugin Architecture

The proposed Unity plugin comprises four modules: Presentation, Simulation Core, Algorithm Library, and Decision Engine. This modular structure separates user interaction, scene execution, native broad-phase implementations, and adaptive selection logic. Figure 1 shows the main data and control flows. The Presentation module sends configuration commands to the Simulation Core, while the Simulation Core submits collision queries to the native Algorithm Library. The Decision Engine analyzes runtime telemetry and issues switching requests. During execution, the Simulation Core sends updated AABB queries to the Algorithm Library, and the Decision Engine forwards performance data to the Presentation module for visualization and export.

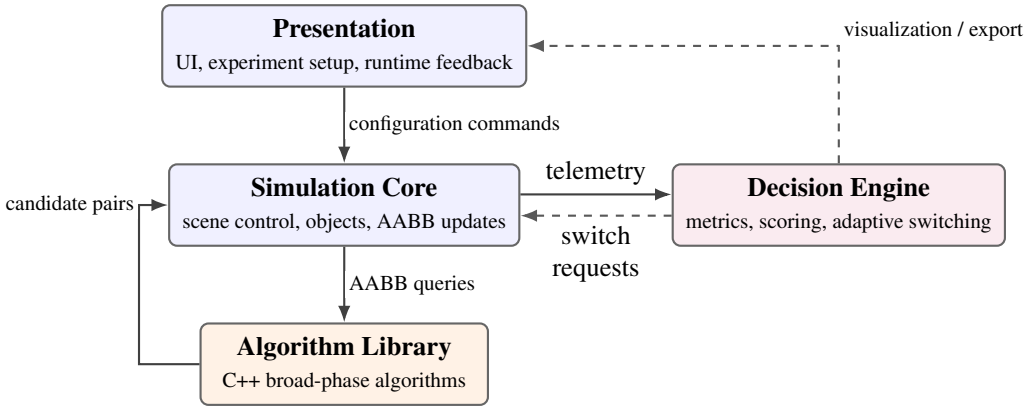

Figure 1. Overview of the proposed Unity plugin architecture.

The plugin[1] supports two operation modes. In Manual mode, the selected broad-phase algorithm remains fixed, supporting controlled experiments. In AI Decides mode, the Decision Engine controls algorithm activation, applies Thompson Sampling, and uses cooldown and post-switch settling to avoid decisions based on transient measurements or warm-up effects. Both modes configure the Simulation Core. Manual mode enables isolated evaluation of each broad-phase algorithm.

6. Experimental Methodology and Test Environment

We describe the protocol used to evaluate the adaptive Broad-Phase selection mechanism implemented. The evaluation was designed as a controlled observational study: for each scenario/object-count condition, the system runs autonomously for a fixed duration, while the decision engine selects broad-phase algorithms using Thompson Sampling. At the end of each run, we record the number of switches, the final posterior values for all candidate algorithms, and the dominant algorithm, defined as the one with the longest active time. The experiments used the Unity plugin connected to the native broad-phase backend in Figure 1. All runs shared the same software configuration and decision parameters on a desktop with an AMD Ryzen 7 5700G CPU at 3.8 GHz and 16 GB of RAM. Logs recorded algorithm activations, switches, per-frame metrics, and final posterior values for offline analysis of convergence and stability. Figure 2 summarizes the adaptive loop: runtime metrics from the active algorithm are converted into a reward, used to update its Beta posterior, and sampled by Thompson Sampling to select the next candidate. The reward reflects end-to-end runtime performance, including broad-phase execution and integration overheads, rather than the isolated cost of the selected algorithm. A switch occurs only when cooldown, settling, and bail-out safeguards allow it.

6.1. Scenario Selection

The plugin provides five simulation scenarios: Free Fall, Brownian Motion, Rotating Gravity, Random Gravity, and Hurricane. We focus on the three scenarios that preserve dynamic and non-uniform workloads throughout execution as shown in Table 2. Free Fall and Random Gravity were excluded from the main evaluation because, after a short

[1]The plugin implementation and experimental artifacts are available in a public repository to support reproducibility: `https://github.com/RaulCF021/Broadmark4unity-v2`.

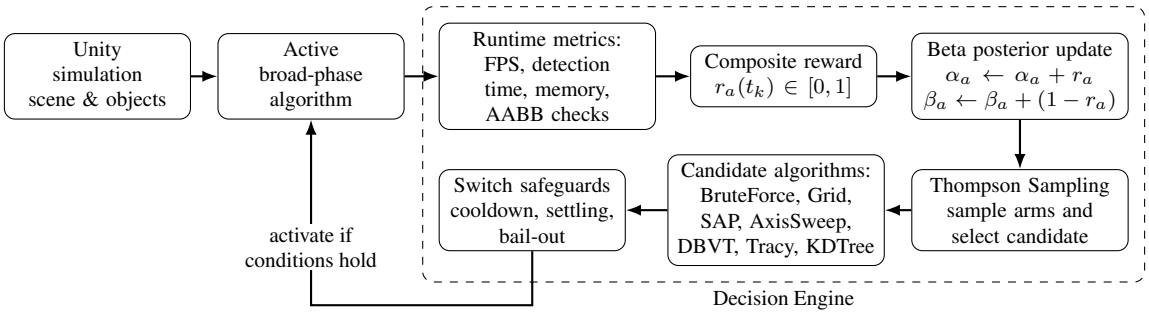

Figure 2. Adaptive broad-phase selection loop based on Thompson Sampling.

transient, they tend to become more uniform or nearly static, offering fewer meaningful regime changes for an adaptive selector.

Table 2. Evaluation scenarios used in the experiments.

Scenario	Motion pattern	Purpose
Brownian Motion	Independent pseudo-random perturbations at each frame.	Spatially diffuse and constantly changing distribution of heterogeneous objects, useful for comparing locality-aware and more uniform strategies.
Rotating Gravity	Gravity direction changes cyclically across four directions.	Non-stationary clustering of heterogeneous objects, transient cost spikes, and repeated workload shifts.
Hurricane	Objects orbit a central point at high speed.	Dense, coherent, high-speed workload with heterogeneous AABBs, used to test convergence under strong geometric structure.

6.2. Experimental Variables and Outputs

Each selected scenario was evaluated with 1,000, 2,000, and 3,000 dynamic objects, covering moderate to heavier workloads. This allows us to observe whether the selector maintains the same preference as the scene scales or shifts toward a different spatial data structure. Objects were randomly generated with heterogeneous sizes and spatial extents, therefore producing AABBs of different dimensions across the simulation. This makes the workload less idealized than uniform-object benchmarks and exposes the selector to variations in object size, distribution, and overlap density.

Each scenario/object-count condition was executed five independent times. This allows us to analyze not only the dominant algorithm in a single run, but also the stability of the selector across repeated executions. We therefore report dominance frequency across the five runs and aggregate statistics for the most frequent dominant algorithm in each condition. For each run, we record the dominant algorithm, post-exploration switches, active time, and final Beta posterior values used by Thompson Sampling.

We report the posterior mean $\mu_i = \alpha_i/(\alpha_i + \beta_i)$ and accumulated evidence $n_i = \alpha_i + \beta_i$; μ^* and n^* refer to the dominant algorithm. The five-run analysis includes dominance frequency and aggregate statistics for active time, μ^*, and n^*.

6.3. Robustness Mechanisms and Execution Procedure

The selector includes two safeguards against pathological exploration. Adaptive FPS weighting increases the FPS contribution to the reward when the frame rate falls below

25 FPS, preventing algorithms with poor visual responsiveness from being favored due to secondary metrics. The real-time bail-out mechanism aborts an arm if a frame exceeds five seconds and assigns ten zero-reward observations, allowing the selector to recover without waiting for the normal cooldown. Each run follows a fixed protocol. The scene is configured with the selected scenario and object count, and the system is set to *AI Decides* mode. An initial exploration phase executes all candidate algorithms with warm-up and measurement intervals. The simulation then runs for a 15-minute observation window, using a decision interval of $\Delta t = 2$ s, a minimum switch cooldown of 40 s, and a bail-out threshold of 5 seconds per frame. Each scenario/object-count combination is treated as an independent experimental cell, with no state or prior shared across cells.

7. Results and Discussion

The analysis focuses on convergence behavior and posterior confidence, rather than on statistically claiming superiority over all fixed-algorithm baselines. Table 3 summarizes the dominant algorithms observed across five independent runs for each scenario/object-count condition. The notation $x/5$ indicates how many times an algorithm became dominant across the five runs of the same condition. Table 4 reports aggregate statistics for the most frequent dominant algorithm in each condition.

Table 3. Dominant algorithms across five runs per scenario and object count.

Scenario	Objects	Dominance frequency	Interpretation
Brownian	1,000	AxisSweep 5/5	Stable sweep-based preference.
Brownian	2,000	AxisSweep 5/5	Stable sweep-based preference.
Brownian	3,000	KDTree 3/5; AxisSweep 2/5	Transitional workload between sweeping and hierarchy.
Rotating Grav	1,000	SAP 3/5; AxisSweep 2/5	Ambiguous low-density cyclic workload.
Rotating Grav	2,000	AxisSweep 5/5	Stable sweep-based preference under cyclic motion.
Rotating Grav	3,000	KDTree 5/5	Stable hierarchical preference under dense cyclic clustering.
Hurricane	1,000	AxisSweep 5/5	Stable sweep-based preference.
Hurricane	2,000	AxisSweep 5/5	Strong stable sweep-based preference.
Hurricane	3,000	KDTree 5/5	Stable hierarchical preference at high density.

Table 4. Thompson Sampling over five runs: dominance frequency, mean switches, and dominant-run averages for active time, μ^*, and n^*.

Scenario	Obj.	Dominant	Freq.	Switches	Active (%)	μ^* / n^*
Brownian	1,000	AxisSweep	5 / 5	20.2±0.8	45.5±6.3	0.977±0.007 / 196.4±27.5
Brownian	2,000	AxisSweep	5 / 5	12.2±2.8	67.8±7.0	0.967±0.004 / 295.8±32.3
Brownian	3,000	KDTree	3 / 5	16.0±2.1	52.9±6.9	0.900±0.023 / 203.0±35.5
Rotating Grav	1,000	SAP	3 / 5	20.0±1.2	42.3±2.9	0.905±0.006 / 179.0±13.0
Rotating Grav	2,000	AxisSweep	5 / 5	9.8±1.5	60.4±8.8	0.807±0.113 / 164.4±35.3
Rotating Grav	3,000	KDTree	5 / 5	11.2±1.5	44.7±9.2	0.244±0.008 / 63.6±16.3
Hurricane	1,000	AxisSweep	5 / 5	21.0±0.0	41.2±2.2	0.970±0.006 / 176.2±9.0
Hurricane	2,000	AxisSweep	5 / 5	6.6±1.7	82.5±3.9	0.954±0.004 / 355.8±21.3
Hurricane	3,000	KDTree	5 / 5	8.8±1.8	72.0±6.1	0.831±0.025 / 245.0±40.7

Across the five independent runs, the selector showed stable behavior in most scenario/object-count conditions. AxisSweep was consistently dominant in Brownian Motion with 1,000 and 2,000 objects and in Hurricane with 1,000 and 2,000 objects. This suggests that sweep-based processing benefits from diffuse or coherent motion when the workload remains within a moderate density range. Hurricane with 3,000 objects consistently shifted to KDTree, indicating that the dense orbital structure favors hierarchical spatial partitioning at higher scale.

Brownian Motion with 3,000 objects behaved as a transitional case: KDTree became dominant in three runs, while AxisSweep dominated in two. This suggests a regime where both temporal coherence and hierarchical spatial partitioning can become competitive. Rotating Gravity was the most ambiguous scenario. At 1,000 objects, dominance alternated between SAP and AxisSweep, while at 2,000 objects AxisSweep was consistently dominant but with higher variability in posterior confidence. At 3,000 objects, KDTree became dominant in all runs, although the posterior mean remained low. This indicates that the selector consistently identified KDTree as the best available candidate under degradation, rather than as a clearly high quality solution. Accordingly, switch counts alone do not characterize selector stability and should be interpreted jointly with active time, posterior mean, and accumulated evidence.

The aggregated Beta posteriors reinforce this interpretation. Stable cases—Brownian 1,000/2,000 and Hurricane 1,000/2,000—show high posterior values for the dominant algorithm, indicating consistent confidence in the selected arm. By contrast, Rotating Gravity 3,000 shows KDTree dominance in all five runs but a low posterior mean ($\mu^* = 0.244 \pm 0.008$), suggesting convergence to the least degraded candidate rather than a clearly high-quality solution.

7.1. Implications for Game Physics

The results suggest that adaptive broad-phase selection is most useful when runtime workloads change spatial coherence, clustering, and bounding-volume overlap. Sweep-based methods were consistently favored in Brownian Motion and Hurricane up to 2,000 objects, while KDTree dominated Hurricane and Rotating Gravity with 3,000 objects. Brownian 3,000 behaved as a transition region, and Rotating Gravity exposed the strongest ambiguity due to cyclic clustering and repeated workload shifts. Heterogeneous AABBs reinforce this interpretation, as the selector must adapt to object count, motion pattern, bounding-volume size, and overlap density. For game physics systems, this suggests that an adaptive selector should identify not only a preferred algorithm, but also when no algorithm is clearly dominant. Stable high-confidence posteriors support exploitation, whereas low-confidence posteriors may indicate the need for fallback policies, scene simplification, or additional candidate algorithms.

7.2. Limitations

This study has limitations. Although each scenario/object-count condition was executed five times, the evaluation is limited to one hardware profile and a controlled set of synthetic workloads. The five runs provide initial evidence of consistency but are insufficient for definitive statistical conclusions. Hardware-specific effects may influence FPS, memory behavior, dynamic-library warm-up costs, and switching overhead. The current evaluation does not isolate the overhead associated with the selector, backend

reconstruction, Unity–native synchronization, and steady-state broad-phase execution. The reward weights were defined according to domain priorities and may require recalibration for other genres or hardware profiles. Although the experiments use randomly generated heterogeneous objects, they may not cover the full diversity of object shapes, lifecycles, constraints, spawning patterns, or interactions found in complete games. Finally, the study does not yet include a comprehensive statistical comparison with fixed-algorithm and heuristic baselines.

8. Conclusions and Future Work

This paper presented a Thompson-Sampling-based mechanism for runtime selection of broad-phase collision detection algorithms in Unity. The system was evaluated across three motion scenarios, three object-count scales, heterogeneous dynamic objects, and five independent runs per condition. Results showed that AxisSweep was stable in Brownian Motion with 1,000 and 2,000 objects and in Hurricane with up to 2,000 objects, while KDTree was stable in Hurricane and Rotating Gravity with 3,000 objects. Brownian Motion with 3,000 objects and Rotating Gravity with 1,000 objects exhibited transitional or ambiguous behavior, showing that posterior confidence can help identify uncertain workloads. The robustness mechanisms—adaptive FPS weighting, real-time bail-out, cooldown control, and post-switch stabilization—helped maintain stable learning under transient overload and pathological algorithm choices. Overall, the study supports bandit-based selection as a promising direction for game physics systems in which the most suitable spatial data structure depends on runtime workload characteristics.

Future work will expand the evaluation to additional hardware profiles, fixed-algorithm and heuristic baselines, larger run sets with statistical testing, and more diverse game-like scenarios. Additional baselines should include random selection, threshold-based switching, UCB, epsilon-greedy, and bandit methods that account for switching costs [Bubeck and Cesa-Bianchi 2012, Lattimore and Szepesvári 2020]. We also plan to investigate contextual bandit formulations that use scene descriptors, such as object density, velocity distribution, and clustering, to guide selection before performance degradation is observed.

Acknowledgments

The authors thank Universidade de Fortaleza, GIRA Lab, and the reviewers for their feedback. M.A.F. Rodrigues acknowledges support from CNPq Grant 314776/2023-0.

References

Avril, Q., Gouranton, V., and Arnaldi, B. (2014). Collision Detection: Broad Phase Adaptation from Multi-Core to Multi-GPU Architecture. *Virtual Reality and Broadcasting*, 6(11):1–13.

Bubeck, S. and Cesa-Bianchi, N. (2012). Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learn.*, 5(1):1–122.

Capannini, G. and Larsson, T. (2017). Adaptive collision culling for massive simulations by a parallel and context-aware sweep and prune algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 24(7):2064–2077.

Chapelle, O. and Li, L. (2011). An empirical evaluation of Thompson sampling. In *Advances in NeurIPS*, volume 24, pages 2249–2257.

Coumans, E. (2018). Bullet physics library 2.88. `https://github.com/bulletphysics/bullet3`.

Ericson, C. (2004). *Real-Time Collision Detection*. Morgan Kaufmann.

Feitosa, R. C. (2026). Adaptive Broad-Phase Collision Detection in Unity: A Plugin-Based Framework for AI-Assisted Algorithm Selection. Master's thesis, PPGIA - Universidade de Fortaleza, Fortaleza, CE, Brazil. `https://unifor.br/web/pos-graduacao/mestrado-informatica`.

Kopta, D., Ize, T., Spjut, J., Brunvand, E., Davis, A., and Kensler, A. (2012). Fast, effective BVH updates for animated scenes. In *Proc. of the SIGGRAPH Symposium on I3D*, page 197–204, NY, USA. ACM.

Lattimore, T. and Szepesvári, C. (2020). *Bandit Algorithms*. Cambridge University Press.

Liu, F., Harada, T., Lee, Y., and Kim, Y. (2010). Real-time Collision Culling of a Million Bodies on Graphics Processing Units. *ACM ToG*, 29:154.

Lo, S.-H., Lee, C.-R., Chung, I.-H., and Chung, Y.-C. (2013). Optimizing Pairwise Box Intersection Checking on GPUs for Large-Scale Simulations. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 23:19:1–19:22.

NVIDIA (2019). Physx 4.1. `https://github.com/NVIDIAGameWorks/PhysX`.

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers*, 15:65–118.

Russo, D. J., Van Roy, B., Kazerouni, A., Osband, I., and Wen, Z. (2018). A tutorial on Thompson Sampling. *Foundations and Trends in Machine Learning*, 11(1):1–96.

Serpa, Y. R. and Rodrigues, M. A. F. (2019). Flexible use of temporal and spatial reasoning for fast and scalable CPU broad-phase collision detection using KD-Trees. *Computer Graphics Forum*, 38(1):260–273. `https://doi.org/10.1111/cgf.13529`.

Serpa, Y. R. and Rodrigues, M. A. F. (2020). Broadmark: A testing framework for broad-phase collision detection algorithms. *Computer Graphics Forum*, 39(1):436–449. `https://doi.org/10.1111/cgf.13884`.

Terdiman, P. (2017). Physics Engine Evaluation Lab (PEEL). `https://github.com/Pierre-Terdiman/PEEL`.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3–4):285–294.

Tornede, A., Bengs, V., and Hüllermeier, E. (2022). Machine learning for online algorithm selection under censored feedback. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8417–8424.

Tracy, D., Buss, S., and Woods, B. (2009). Efficient Large-Scale Sweep and Prune Methods with AABB Insertion and Removal. *IEEE Virtual Reality*, pages 191–198.

Woulfe, M. and Manzke, M. (2009). A framework for benchmarking interactive collision detection. In *Proceedings of the 25th SCCG'09*, pages 205–212, NY, USA. ACM.