Proceedings of the 2025 Winter Simulation Conference
E. Azar, A. Djanatliev, A. Harper, C. Kogler, V. Ramamohan, A. Anagnostou, and S. J. E. Taylor, eds.

MULTI-AGENT MARKET SIMULATION FOR DEEP REINFORCEMENT LEARNING WITH HIGH-FREQUENCY HISTORICAL ORDER STREAMS

David Byrd[1]

[1]Dept. of Computer Science, Bowdoin College, Brunswick, ME, USA

ABSTRACT

As artificial intelligence rapidly co-evolves with complex modern systems, new simulation frameworks are needed to explore the potential impacts. In this article, I introduce a novel open source multi-agent financial market simulation powered by raw historical order streams at nanosecond resolution. The simulation is particularly targeted at deep reinforcement learning, but also includes momentum, noise, order book imbalance, and value traders, any number and type of which may simultaneously trade against one another and the historical order stream within the limit order books of the simulated exchange. The simulation includes variable message latency, automatic agent computation delays sampled in real time, and built-in tools for performance logging, statistical analysis, and plotting. I present the simulation features and design, demonstrate the framework on a multipart DeepRL use case with continuous actions and observations, and discuss potential future work.

1 INTRODUCTION

In this article, I introduce minABIDES, a new multi-agent market simulation platform designed for research experiments in intraday or high-frequency trading by deep reinforcement learning algorithms, optionally in the presence of other strategic trading agents. The simulation is deliberately lightweight, containing less than one thousand total lines of program code, but has many useful features built in. I hope it will prove a useful jumping off point for other researchers, or for those who might wish to construct a class assignment. The complete framework code is available at https://github.com/davebyrd/minabides.

Simulation fills an important gap in the financial market literature. It can be used to explore counterfactual scenarios, to augment sparse training data, or to permit investigation of harmful effects in a safe environment. I was motivated to create a new simulation framework to address specific desiderata:

- Driven by historical data
- Supports deep reinforcement learning
- Simulates high-frequency trading
- Research and classroom use
- Single agent backtesting
- Multi-agent interaction
- Built in stats and visualization
- Easy hyperparameter searches

Example research questions minABIDES could help the community answer include: Which of these four deep learning algorithms can best adapt to different time scales? Is the performance advantage of a "smarter" trading algorithm enough to overcome the additional computational delay? Can a learning agent tasked with profit maximization inadvertently learn to manipulate other traders? When not replaying historical data, what combination of simple trading strategies from the financial literature produces the most realistic market evolution?

In the current work, I survey the related literature, introduce the design and key features of my simulation platform, illustrate its ease of use, and show detailed tables and plots from a preliminary set of experiments in high-frequency deep reinforcement learning against historical data, both with and without the presence of other trading agents.

2 RELATED WORK

Simulations of complex, real-world systems tend to fall into one of three paradigms. Continuous simulations, for example of physics, rely on mathematical processes and differential equations to estimate the state of the system at any arbitrary point in time (Cellier and Kofman 2006). Such simulations can be very computationally efficient, but not every system lends itself to the underlying processes. Stepwise simulations, for example of a turn-based game, advance time in fixed increments, polling each participant for a possible action at every time step (Mnih et al. 2013). These are structurally the simplest systems, but inefficient due to polling overhead for systems in which the participant activity is sparse. Discrete event simulation, for example of customer arrival and service at a fast food restaurant, is organized around a priority queue of events sorted by increasing event time (Goldsman and Goldsman 2015). This can require a more complex design, because participants must be activated by events in the queue at irregular time intervals, but are efficient even for sparse activity tasks because the time in which no activity occurs is not simulated at all. My contribution here is a discrete event simulation of financial markets using high-resolution historical data.

2.1 Multi-agent Market Simulation

The minABIDES discrete event simulation (DES) framework is an example of agent-based simulation (ABS) in which separate logical processes or entities make independent, sequential action decisions with only limited knowledge of other participants and their activities. Agent-based modeling has a long history in economics (Tesfatsion 2002) and the social sciences (Axelrod 1997), but has recently surged in popularity as "agentic" models based on tool use by large language models (LLMs), particularly in fields like healthcare (Qiu et al. 2024).

My framework is far from the first agent-based market simulation. Harry Markowitz, Nobel prize winner known for his work in portfolio theory and the efficient frontier, investigated agent-based financial market simulation and found it useful, creating the JLMSim named after the collaborating authors (Jacobs et al. 2004). JLMSim uses trading rules and simple strategies, interacting with an order book, to reproduce the market. It was powerful for its time, but did not permit implementation of complex custom strategies such as machine learning based agents, and its source code was not released. Several other innovative ABS market simulations have been introduced in recent years. MarketSim allows strategic agents to adapt select hyperparameters in a game-theoretic manner to Nash-equilibrate a simulated market (Wellman and Wah 2017). ABIDES is an open source general-purpose DES with cryptography, economy, market, and other modules (Byrd et al. 2020). In its market module, it provides various fixed-policy agents and a simple tabular Q-learning agent with discrete observations of the market. PyMarketSim is an update to the earlier work that adds a reinforcement learning-based background agent (Mascioli et al. 2024).

As the name suggests, minABIDES is most closely related to the ABIDES framework, but with many important differences. It is a fresh implementation from the ground up, without reference to ABIDES' source code, to best incorporate years of lessons learned. minABIDES is focused only on market simulation, with a design principle to be as minimal and elegant as possible. The complete simulation is less than one thousand lines of code, compared with tens of thousands for ABIDES, while still supporting key features not found in its predecessor. Each of MarketSim, ABIDES, and PyMarketSim is driven by mean-reverting random processes, while minABIDES ingests the complete historical order stream from pre-market through close for each simulated date. The data is provided by LOBSTER (Huang and Polak 2011) under academic license, with the original data source being Nasdaq TotalView. Further distinctions are drawn in the section on design.

2.2 Other Market Simulation

It is also possible to evaluate strategies or learning agents in a commercial platform or using general purpose open source simulation tools. In this section, I list several of significance and contrast them with minABIDES.

Common commercial market simulation platforms like MetaTrader, OneTick, and QuantConnect operate with included or user-provided data to backtest a single strategic agent against non-adaptive historical data. These platforms each provide access to classification and regression machine learning algorithms. minABIDES supports this style of simple backtesting by adding only an exchange and a single strategic agent to a simulation. However, minABIDES also allows adding an arbitrary number of other strategic agents to produce a dynamic market which reacts to every agent-initiated order, naturally deviating from the historical data when appropriate. The minABIDES focus on deep reinforcement learning, and its inclusion of functional trading agents for three common algorithms, sets its machine learning capabilities apart by allowing an agent to directly learn the potential impact of its own actions on the future state of the market.

A general purpose simulation framework can be a good starting point for the development of a market simulation. SimPy is a Python-based discrete event simulation based around resources, queues to utilize those resources, and processes which flow between resources (SimPy 2002). It is widely used for logistics, services, and manufacturing simulations. MASON supports the continuous or discrete event simulation of a large number of agents in Java and provides real-time visualization for "physical" environments like robotics or social simulations (Luke et al. 2005). AnyLogic (free or commercial) supports graphical modeling of agent-based modeling in Java, especially for system dynamics, supply chain, or transportation simulation (Borshchev 2014). SimPy, MASON, and minABIDES are open source, while AnyLogic is not. The primary distinction between minABIDES and these other platforms is a batteries-included focus on market simulation, trading off multi-domain flexibility for ease of implementation within the target domain. The ABIDES framework, predecessor to minABIDES, is more general purpose, having been used for simulation of macro-economic processes and cryptographic protocols, among other applications.

2.3 Reinforcement Learning

A Markov decision process (MDP) describes an environment which transitions between states based on both agent actions and random probabilities, in which an agent attempts to maximize the rewards it can obtain from some transitions (Bellman 1957). It is usually described through variables S, A, T, R representing a set of states, actions, probabilistic state transitions $Pr(s'|s,a)$, and rewards $R(s,a,s')$. When the entire problem, including all listed variables, is known a priori, it can be efficiently solved using a dynamic programming algorithm. When the MDP parameterization is not known, we encounter the reinforcement learning (RL) problem (Sutton et al. 1998), in which an agent must explore the environment to estimate the parameters of the MDP while simultaneously attempting to maximize its total reward.

Solutions to the RL problem fall in two primary categories: model-based, in which the agent tries to form an explicit model of the world (especially T, R), and model-free, in which the agent more directly learns which actions are best to take from which states. The basis of modern model-free approaches to RL is Q-learning (Watkins and Dayan 1992), in which a version of Bellman's equation is iteratively optimized:

$$Q^{\pi}(s,a) = R_s(a) + \gamma \sum_{s'} Pr_{ss'}[\pi(s)] V^{\pi}(s') \tag{1}$$

In the above equation, most variables correspond directly to the underlying MDP. γ is a discount factor that makes future rewards increasingly less desirable relative to immediate rewards. π refers to the policy of taking certain actions from states. V is a value function representing the utility of a state to the agent's future prospects. The Q-value of a state-action pair is the estimated total discounted utility of taking the action from the state and then continuing to follow the current policy.

Traditional Q-learning is tabular, requiring a finite set of discrete states and actions. If the agent cannot observe all states during training due to a large or continuous space, arbitrary actions will result. If the space is quantized, the agent can no longer learn different actions for states which were initially distinct.

The solution to this problem was deep reinforcement learning (DeepRL), which avoids the dimensionality issues of prior approaches. The first, and still most popular, approach to DeepRL with discrete action selections was deep Q-network (DQN), in which the substitution of a neural network for a tabular array is straightforward (Mnih et al. 2013). The input neurons of the network receive continuous state observations and each output neuron is considered the Q-value estimate of a distinct action. The network is trained with a loss function very similar to the Bellman equation, thus improving its Q-value estimates over time:

$$\left(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)^2 \tag{2}$$

which is simply the squared difference between the Q-value estimated from the successor observation with the predicted Q-value.

A major successor to DQN is the deep deterministic policy gradient (DDPG), which also permits continuous actions (Lillicrap et al. 2015). DDPG is also an example of an actor-critic algorithm, in which an actor network transforms an input state representation into a proposed set of real-valued actions, and a critic network transforms the combined state-action into an estimated Q-value. An important refinement to DDPG is twin-delayed DDPG (TD3), which estimates utility as the minimum of two critic networks, and updates only the critics at every time step (Fujimoto et al. 2018). After the actor selects a continuous set of actions plus normally distributed noise (to better explore the state-action space), it is updated in mini-batches using:

$$y = r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a}) \tag{3}$$

These changes stabilize the learning process and help prevent the optimism bias common in both tabular and deep Q-Learning.

The minABIDES framework includes all three of these DeepRL algorithms as trading agents: DQN, DDPG, and TD3. Their incorporation to the simulation is further discussed in the section on design.

2.4 Reinforcement Learning in Multi-agent Market Simulation

Learning agents have long been used in financial market trading and simulation, but over the last several years, there has been interest in the application of specifically RL-based agents to financial market simulations which are interpreted to have an underlying MDP representing market state transitions and potential rewards if the agent can select the proper actions at the right times. Simulation is particularly important in this area to understand how learning-based trading agents might display unexpected or even malicious behavior, as one would otherwise have to risk unleashing such behaviors in the world, or else wait for a natural experiment, at which point damage may have already been done. Along these lines, one investigator has found that a tabular Q-Learning agent, if permitted an order cancellation action, will learn to increase its profit by spoofing (manipulating demand signals with fake orders) other agents even if not instructed to do so (Byrd 2022). Another group explores the relationship between market liquidity and the prevalence and profitability of spoofing learned by an RL-based agent (Gu et al. 2024). Researchers have also recently demonstrated that a DDPG-based trader can effectively learn to control the sentiment of LLM-based social media posts it generates as a method of manipulating another sentiment-aware trader's perception of an asset's value (Byrd 2025). The minABIDES simulation framework aims to facilitate faster, further progress along these important lines of inquiry.

3 SIMULATION DESIGN

The design of minABIDES is intentionally straightforward to ease its introduction to new research groups or student classroom activities. There are four files that control the main simulation, five files containing agents, one utility file, and one file that computes statistics and draws plots from system logs.

3.1 Framework Overview

All simulations begin with `run_exp`. The only required parameter is the name of an experiment to run, which must match a program in the experiments directory. However, `run_exp` also accepts a range of simulation parameters that are relevant to every experiment, including start and stop dates and times, market open and close times, how many order book levels to offer agents in their observations, and much more. `run_exp` hands off control to the requested experiment.

There is a comprehensive example experiment in `experiments/rl` which populates a market with configurable background and learning agents, and is a good starting place to design one's own configurations. The experiment file should add additional command-line parameters specific to the experiment, instantiate and configure all desired agent participants, and then turn control over to the main simulation engine to handle all subsequent activity.

The `simulation` module contains the core logic of the framework. The high level `run_experiment` method understands how to guide the agents in multiple trips through multiple days of training, followed by validation and testing. The lower level `simulate` method is responsible for a single market period (one trip through a day or partial day), and handles message processing, timekeeping, and logging.

A few additional files are needed to handle the flow, recording, and transaction of orders, as well as directing historical orders into the simulation:

- The `history` module used by `simulation` to read historical orders and inject them to the simulation event queue for processing at the correct time. Its `reconstruct` method builds an reality-matching order book to start interactive simulation at any desired time. The `fast_forward` method uses a cached version of that book to provide future simulations with the same initial state in a fraction of the time.
- The `exchange` agent must be present in every experiment. It maintains limit order books for one or more stock symbols, is capable of processing requests to place, reduce, or cancel orders, and provides order book observations to other agents at a frequency of their choosing.
- The `orders` module contains an `Order` dataclass and an `OrderBook` which knows how to enter, remove, adjust, or execute orders while maintaining an accurate record of its current state. `OrderBook` also supports a Nasdaq-like "good until" field for auto-expiring orders. For efficiency, it incrementally maintains a configurable sequence of periodic limit order book snapshots for the exchange to provide to trading agents at no additional computational cost.

The above files are all that is required for the logic flow of a basic simulation experiment. Together they can produce a historically-accurate transaction log and limit order book log at any desired level of depth for any time period for any symbol. In most cases, however, the researcher will want to include one or more trading agents to interact with each other, as well as the historical order flow. The currently included "background" (non-learning) agents in the `agents/market` module are:

- The `Market Maker` agent provides liquidity by maintaining a limit order ladder on both sides of the current mid-price, reducing volatility and the likelihood of having no offers for an asset.
- The `Momentum` agent follows a basic momentum strategy, one of the few known exceptions to the idea of efficient markets. It considers the change in price over a period of time and trades on the assumption that the trend will continue.
- The `Noise` agent places randomly priced limit orders within a range around the current mid-price. It represents uninformed market participants.
- The `Order Book Imbalance` agent computes a depth-weighted ratio of the share-volume supply and demand in the current order book. It enters the market on the assumption that a high imbalance will resolve with price movement in the direction of lesser demand.

- The `Value` agent noisily observes an extrinsic *fundamental value* series for its traded asset, and trades on the assumption that market prices will revert to the fundamental. In so doing, it helps keep the simulated market somewhat in line with history.

The framework also provides a `util` module for common tasks, a `stats` module for automatic descriptive statistics and plots, and an example shell script to manage large, parallel experimental trials on a slurm-like high performance computing cluster (HPC).

3.2 Framework Features

The minABIDES framework draws inspiration from prior works in the area of ABS market research:

- `simulate` maintains a global virtual time (GVT) and an individual "current time" for each agent, which may be in the past or future of GVT. When it is time to direct a message to an agent, the agent will fast-forward to GVT if it was in the past, or the message will be delayed if the agent is still in the future (i.e. busy from a prior action).
- The framework contains a variable network latency calculation that allows each agent to have a different, configurable minimum latency to the exchange, to which additional stochastic delivery delays are added.
- The `market` package contains featureful base agent classes which handle messages, performance reporting, portfolio tracking, marking to market, order submission, and more. Prior works require agent reinstantiation each market period, presenting problems for stateful learning agents. minABIDES gracefully handles start of day logic to surgically retain or reset agent state as needed.

It also adds several new key features:

- The system relies on a nanosecond-precision historical data stream which contains every order, at any price level, for a given symbol on a given date within roughly the past eighteen years. Both `simulate` and `exchange` are built to handle this volume of data and level of precision.
- minABIDES uses co-routines, which can incrementally yield multiple return values from a single method call, for all message delivery to agents. This allows the calling method to take action on each yielded result as it becomes available. This is used for automatic real-time computation delay, incrementally delaying each outbound message as the agent executes its logic.
- The history module can reconstruct an accurate order book at any requested point in time. The results are disk-cached, allowing the simulation to fast forward to agent start time without unnecessary overhead in future simulations.
- The order book supports good-until orders that automatically expire at a certain time with no agent action required, resulting in faster simulation compared with order cancellation messages.
- The framework includes an assumption of training learning agents, and supports the idea of separate training, validation, and testing periods out of the box. If there are no learning agents, validation and testing are automatically skipped.
- minABIDES automatically generates detailed machine-readable logs for each individual agent's monetary performance per day, each learning agent's loss values over time, and the progression of the order book and transactions every day. These can be summarized and plotted by the included `stats` module.
- Common RL utilities are included without the requirement for additional library downloads, such as `Box` and `Discrete` action spaces as well as `Replay Buffer` and `Replay Sample` classes. These are based on other open source projects like gymnasium and stable-baselines, and are properly attributed in the project's license file.

Three Deep Reinforcement Learning algorithms form the basis of included trading agents. All three DeepRL agents accept a continuous state space as input, and have configurable hyperparameters set in `experiments/rl` such as learning rate, maximum trade size, maximum position size, sequence processing encoder, network size, gamma discount factor, and many more.

- `rl/dqn` contains a DQN-based agent (see Section 2.3) that generates a discrete action space. The included agent's actions are to buy or sell its configured trade size, or hold its position. Additional actions can be added, such as placing or canceling limit orders. The agent also supports configurable training and target update frequency.
- `rl/ddpg_td3` contains both a DDPG and TD3 agent (see Section 2.3) that generates a continuous action space. Experiments in this work require only one action (trade direction/quantity), but additional actions can be configured to control next desired action time, market sentiment, or other factors. The agent also supports configurable exploration noise, policy noise, and policy update frequency.

4 IMPLEMENTATION NOTES

The minABIDES framework is implemented in pure Python 3.12 and is dependent on just four common libraries: matplotlib, numpy, pandas, and torch. No complicated installation or niche modules are required. In performance tests, the system processes about five thousand messages (orders/notifications to/from the exchange) per second. Even with DeepRL agents, high frequency historical data, and a large order book, minABIDES usually has a peak memory utilization under 500 MB for an experiment.

Full source code is available at https://github.com/davebyrd/minabides and omitted here in the interest of brevity. For two examples of the simplicity of the implementation, consider the message delivery loop in `simulation.simulate` and the configuration logic in `experiments/background`, which consist of seven and nine lines of straightforward code, respectively.

As the simulation progresses, log files of agent performance, learning loss, and order book progression are automatically generated and can be visualized with the included `stats` module.

5 EXPERIMENT: DEEP REINFORCEMENT LEARNING

To demonstrate minABIDES' potential for the area of DeepRL in historically-driven multi-agent financial market simulation, I now present a preliminary experiment investigating whether the included TD3-based agent can successfully learn an intraday trading task. The trader is evaluated both alone and in the presence of other agents. Its hyperparameter search results are also carried forward several years to see if the results still hold.

5.1 Historical Replay

It is not technically necessary to have *any* agents in the simulation other than the exchange. In this case, the simulation will successfully recreate the exact limit order book and transaction history of the real market day for which data is provided. We can then use the provided `stats` module of minABIDES to visualize the resulting order book. See Figure 1 for reconstructed order books with different levels of depth at very different snapshot frequencies.

5.2 DeepRL Backtest

The minABIDES framework makes it easy to run parallel hyperparameter searches for DeepRL trading agents on a computational cluster. This section presents the results of such a search for a specific date in 2019, the search space for which is summarized in Table 1. Other parameters are at default values as listed in the RL experimental configuration file in the linked repository. Although due to computational and time constraints, I consider just one day in this preliminary experiment, it is still a useful test: the learner is

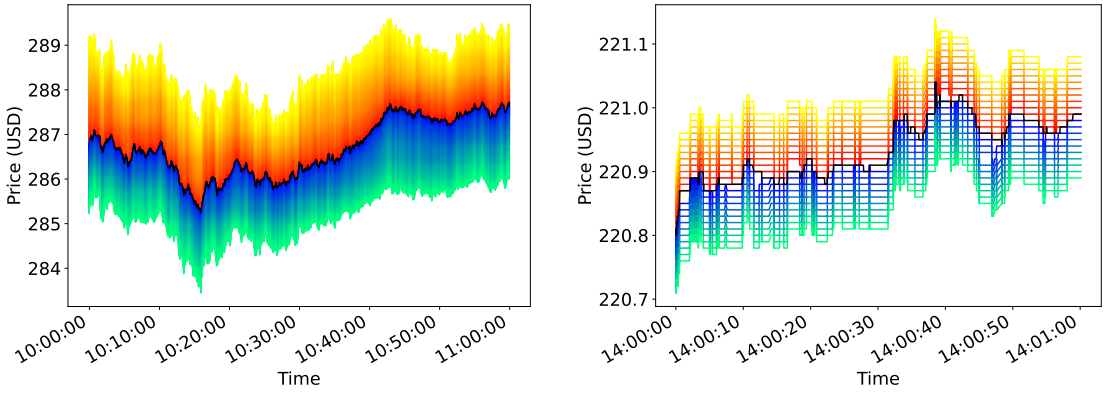

Figure 1: Historical limit order book showing 100 price levels over one hour with a five second snapshot frequency (left) and 10 price levels over one minute with one millisecond snapshot frequency (right).

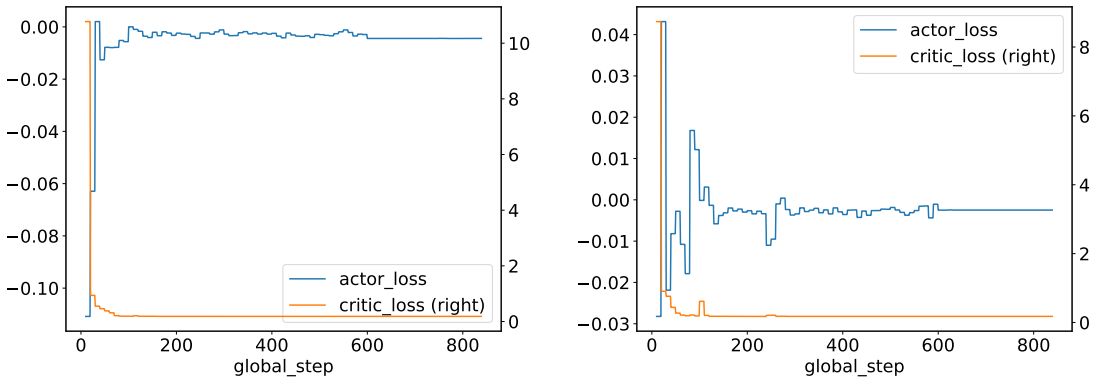

Figure 2: Training loss curves for TD3 alone (left) and TD3 with background agents (right).

trained to trade on a time period early in the day, the best trained learner is selected using a validation period later in the day, and that learner is evaluated on an out of sample period still later in the day. Thus, all validation and test data is in the future of all training data.

In all current experiments, each observation of the TD3 agent is its current inventory of holdings and open orders, plus a normalized sequence of the twenty most recent depth-10 limit order book snapshots at five second intervals. The single continuous output action is interpreted as a real-valued fraction of the agent's maximum inventory which should be bought or sold. The agent receives scaled rewards based on the periodic change in its portfolio value as marked to market.

The experiment also serves as a useful demonstration of the custom `stats` module included with minABIDES. With a single simple command line, it produced all of the displayed tables and figures in this paper directly from the simulation logs. Table 2 and Figure 2 (left side) show the TD3 agent performance and training loss curve. The TD3 algorithm converges to near-zero actor and critic loss as expected. The rapid drop in critic loss is reasonable given the small scale of the rewards (portfolio value change during high frequency trading). The agent adapts to the high-frequency market data, becoming profitable in the median case both in and out of sample. Its performance is near zero in the mean case due to the negative outlier periods. This is common with learning trading agents: they perform well in the median, but mean performance is harmed by days in which it performs particularly badly.

Table 1: Hyperparameter search space for TD3 learning agent.

Hyperparameter	Values	Hyperparameter	Values
Exploration Noise	0.2, 0.5	Embed size	3, 6
Policy Noise	0.2, 0.5	Encoder	lstm, none
Policy Frequency	2, 10	Learning Rate	$5e^{-1}, 5e^{-2}, 5e^{-3}, 5e^{-4}$

Table 2: Descriptive statistics for 480 experiments on 2019-12-30 with the TD3 learning agent. Results based on dollar profit per trading period.

Mode	Mean	Std	Min	25%	50%	75%	Max
test_is	-2.92	109.59	-256.62	-111.13	77.34	105.39	207.04
test_oos	0.42	394.33	-542.62	-396.74	356.53	391.39	493.04
test_val	-0.95	273.08	-421.12	-275.24	238.65	269.89	371.54
train	-2.59	109.55	-256.62	-111.08	76.88	105.65	207.04

5.3 Multi-agent Interactive Backtest

In this section, varying quantities of background agents (momentum, noise, order book imbalance, and value) are introduced to trade interactively against one another and the historical data. In the literature, all of these except the noise agent tend to perform well against a mean-reverting market process. Unsurprisingly, they do not perform well against historical data, as this Nasdaq high-frequency order stream contains many sophisticated real-world strategies from investment banks, hedge funds, fintech firms, and more.

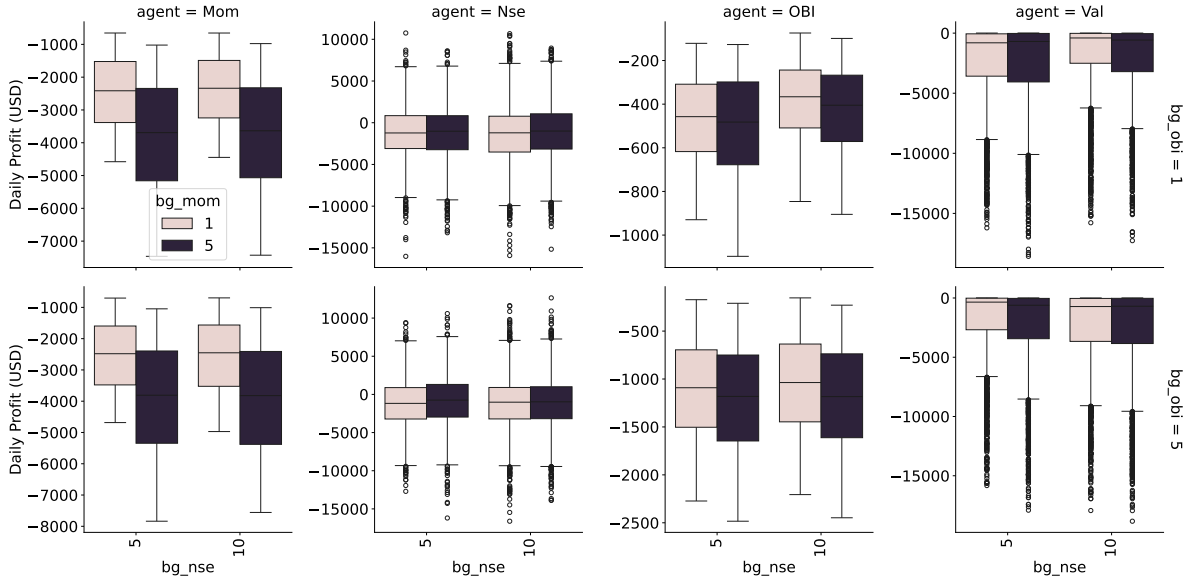

Figure 3: Daily performance over 480 simulations by agent type (column), and the number of order book imbalance agents (row), momentum agents (hue), and noise agents (x-axis of subplots).

The distribution plot in Figure 3 is another output of the included custom `stats` module in minABIDES. Despite the background agents' poor performance, we can see several expected properties emerge:

- As more agents attempt to pursue the same strategy, it performs worse on a per-agent basis.
- The order book imbalance agent has the highest return and the lowest variance of returns. It is the most sophisticated strategy.

- The number of noise agents only really matters to the momentum agent, as it dampens the directional signals being observed.
- The noise agent has a very wide dispersion of results, centered slightly below zero, and quite symmetric to the upside and downside. This tracks for an agent placing random orders.
- The value agent is little affected by the presence or quantity of any other agent. This is because it receives extrinsic observations of a "true value" for the stock.

While the unintelligent agents all lose money in the mean against historical data replay, they display important qualities and show that the simulation is sensible.

5.4 Complete Multi-agent Simulation

Here I combine the TD3 agent with the background agents, primarily to assess whether the TD3 agent can adapt to the presence of other agents, rather than simply learning the historical data. The experiment spans 2,400 training periods and 480 of each test period type. We would expect the sophisticated TD3 learning agent to substantially outperform the unintelligent background agents. In Figure 4, we see exactly that, again noting some important observations:

- The TD3 agent has slightly positive results in the mean and median case despite the extra confusion of the background agents.
- TD3 has substantially better performance than any of the other agents.
- All of the unintelligent agents do best while TD3 is still learning (mode = train), then worse once TD3 has learned the market.

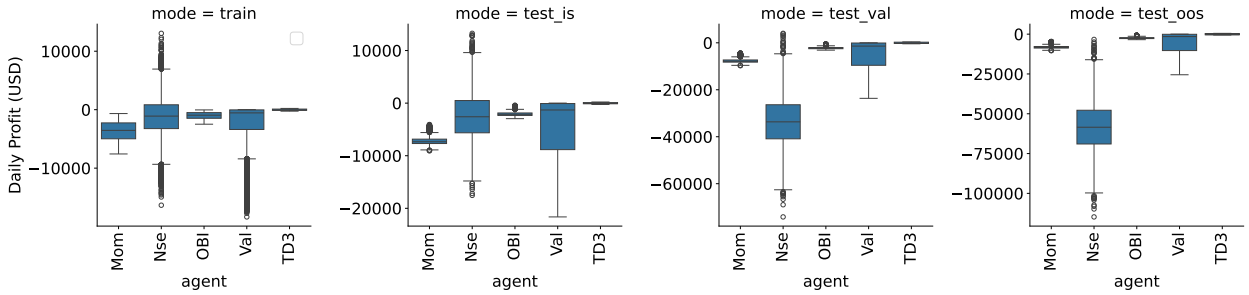

Figure 4: Daily performance by training mode (column) and agent type (x-axis of subplots).

From the right subfigure of previous Figure 2, we can again observe that both the actor and critic networks of the TD3 algorithm converge as expected, even in the presence of other market participants.

5.5 Forward Testing DeepRL

As a final test, I bring the TD3 agent forward to a series of five dates in 2024, and train it using the same hyperparameters which were obtained from the 2019 data. Without new hyperparameter selection, this should show whether the agent sensitivity to hyperparameters over time is low enough to fit itself to current market conditions without another expensive search. Table 3 shows that the agent can learn the future market using the prior hyperparameters. Four of the five tested dates are positive in both the mean and median case, and three days have excellent annualized Sharpe Ratios. Figure 5 shows the converging learning loss and the order book progression.

Through each of the example experiments, the participating agents produced sensible results consistent with expectations and the literature, giving confidence in the reasonableness of the market simulation.

Table 3: Descriptive statistics on dollar profit per period for 80 experiments (TD3 agent, out of sample, five days). Annualized Sharpe Ratio (ASR) is also shown.

Date	Mean	Std	ASR	Min	25%	50%	75%	Max
2024-09-03	30.60	242.70	7.22	-250.08	-245.23	217.12	245.17	253.39
2024-09-04	14.51	99.88	8.31	-131.68	-87.64	72.71	98.28	123.87
2024-09-05	-67.58	281.51	-13.74	-303.06	-284.43	-254.41	274.15	298.70
2024-09-09	3.78	34.86	6.21	-48.79	-27.89	19.93	27.68	46.87
2024-09-10	0.23	55.03	0.24	-96.63	-32.15	1.52	23.22	124.40

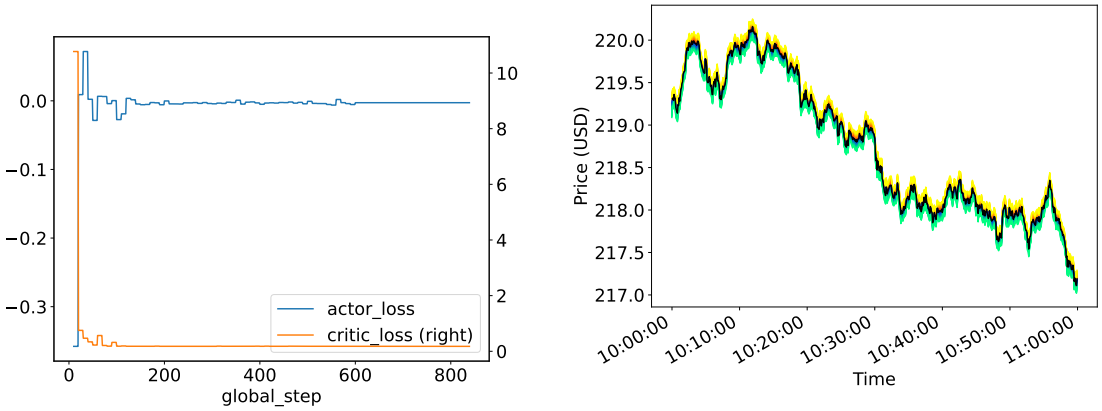

Figure 5: TD3 in forward testing, showing loss curve (left) and depth 10 order book progression (right).

6 CONCLUSION AND FUTURE WORK

Herein I have presented minABIDES, a new multi-agent market simulation framework centered on historical high-frequency data and built for deep reinforcement learning (DeepRL) research. The framework is open source under a permissive license, and should be suitable for either research or classroom use due to its small size and straightforward design. I illustrated the utility of the system by showing a hyperparameter search on a DeepRL trader that then learns to trade the market either alone or in the presence of other non-learning agents. The trader's low hyperparameter sensitivity was demonstrated by retraining on five days of new data that are five years in the future from the hyperparameter selection process.

These proof-of-concept simulation experiments focus on a single deterministic learning agent and simple background traders. They exclude important algorithms and strategies not yet supported by the minABIDES simulation, including proximal policy optimization (PPO), adaptive market making, self-equilibrating parameterized non-learning agents, and multi-agent reinforcement learning (MARL) approaches. These, along with the generality of my approach to other assets, asset classes, and market conditions, are left as future work.

Thus far, minABIDES has been used to study risks and remedies around the introduction of advanced AI to socio-economic markets and networks, including DeepRL agents spontaneously learning to spoof a market or to manipulate investor sentiment by controlling a large language model. I believe that minABIDES may be particularly useful for such normative learning applications, hope that other researchers will find the platform easy to adapt to their needs, and invite potential collaborators to contact me.

REFERENCES

Axelrod, R. 1997. "Advancing the Art of Simulation in the Social Sciences". In *Simulating Social Phenomena*, edited by R. Conte, R. Hegselmann, and P. Terna, 21–40. Berlin, Heidelberg: Springer Berlin Heidelberg https://doi.org/10.1007/978-3-662-03366-1_2.

Bellman, R. 1957. "A Markovian Decision Process". *Journal of Mathematics and Mechanics* 6(5):679–684.

Borshchev, A. 2014. *Multi-method modelling: AnyLogic*, Chapter 12, 248–279. John Wiley & Sons, Ltd https://doi.org/https://doi.org/10.1002/9781118762745.ch12.

Byrd, D. 2022. "Learning Not to Spoof". In *Proceedings of the Third ACM International Conference on AI in Finance*, ICAIF '22, 139–147. New York, NY, USA: Association for Computing Machinery https://doi.org/10.1145/3533271.3561767.

Byrd, D. 2025. "Exploring Sentiment Manipulation by LLM-Enabled Intelligent Trading Agents". *arXiv preprint arXiv:2502.16343* https://doi.org/10.48550/arXiv.2502.16343.

Byrd, D., M. Hybinette, and T. H. Balch. 2020. "ABIDES: Towards High-Fidelity Multi-Agent Market Simulation". In *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM-PADS '20, 11–22. New York, NY, USA: Association for Computing Machinery https://doi.org/10.1145/3384441.3395986.

Cellier, F. E., and E. Kofman. 2006. *Continuous System Simulation*. Boston, MA: Springer US https://doi.org/10.1007/0-387-30260-3.

Fujimoto, S., H. van Hoof, and D. Meger. 2018, 10–15 Jul. "Addressing Function Approximation Error in Actor-Critic Methods". In *Proceedings of the 35th International Conference on Machine Learning*, edited by J. Dy and A. Krause, Volume 80 of *Proceedings of Machine Learning Research*, 1587–1596: PMLR.

Goldsman, D., and P. Goldsman. 2015. *Discrete-Event Simulation*, 103–109. London: Springer London https://doi.org/10.1007/978-1-4471-5634-5_10.

Gu, A., Y. Wang, C. Mascioli, M. Chakraborty, R. Savani, T. L. Turocy *et al*. 2024. "The Effect of Liquidity on the Spoofability of Financial Markets". In *Proceedings of the 5th ACM International Conference on AI in Finance*, ICAIF '24, 239–247. New York, NY, USA: Association for Computing Machinery https://doi.org/10.1145/3677052.3698634.

Huang, R., and T. Polak. 2011. "LOBSTER: Limit Order Book Reconstruction System". *Available at SSRN* https://doi.org/10.2139/ssrn.1977207.

Jacobs, B. I., K. N. Levy, and H. M. Markowitz. 2004. "Financial Market Simulation". *The Journal of Portfolio Management* 30(5):142–152 https://doi.org/10.3905/jpm.2004.442640.

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, *et al*. 2015. "Continuous Control with Deep Reinforcement Learning". *arXiv preprint arXiv:1509.02971* https://doi.org/10.48550/arXiv.1509.02971.

Luke, S., C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. 2005. "MASON: A Multiagent Simulation Environment". *SIMULATION* 81(7):517–527 https://doi.org/10.1177/0037549705058073.

Mascioli, C., A. Gu, Y. Wang, M. Chakraborty, and M. Wellman. 2024. "A Financial Market Simulation Environment for Trading Agents Using Deep Reinforcement Learning". In *Proceedings of the 5th ACM International Conference on AI in Finance*, ICAIF '24, 117–125. New York, NY, USA: Association for Computing Machinery https://doi.org/10.1145/3677052.3698639.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra *et al*. 2013. "Playing Atari with Deep Reinforcement Learning". *arXiv preprint arXiv:1312.5602* https://doi.org/10.48550/arXiv.1312.5602.

Qiu, J., K. Lam, G. Li, A. Acharya, T. Y. Wong, A. Darzi, *et al*. 2024. "LLM-based Agentic Systems in Medicine and Healthcare". *Nature Machine Intelligence* 6(12):1418–1420 https://doi.org/https://doi.org/10.1038/s42256-024-00944-1.

SimPy 2002. "SimPy: a Process-based Discrete-event Simulation Framework". https://simpy.readthedocs.io. Accessed: 2025-08-22.

Sutton, R. S., A. G. Barto *et al*. 1998. *Reinforcement Learning: An Introduction*, Volume 1. MIT Press Cambridge.

Tesfatsion, L. 2002, March. "Agent-Based Computational Economics: Growing Economies From the Bottom Up". *Artif. Life* 8(1):55–82 https://doi.org/10.1162/106454602753694765.

Watkins, C. J., and P. Dayan. 1992. "Q-learning". *Machine Learning* 8:279–292 https://doi.org/10.1007/BF00992698.

Wellman, M. P., and E. Wah. 2017. "Strategic Agent-Based Modeling of Financial Markets". *RSF: The Russell Sage Foundation Journal of the Social Sciences* 3(1):104–119 https://doi.org/10.7758/RSF.2017.3.1.06.

AUTHOR BIOGRAPHY

DAVID BYRD is the Marvin H. Green Jr. Assistant Professor of Computer Science at Bowdoin College in Brunswick, Maine. His research interests include multi-agent simulation, analysis of complex real-world systems, and artificial intelligence, especially sequential decision making problems such as financial markets and games. He has authored two dozen academic articles on these topics. His most recent line of research explores unintended negative behaviors learned by AI in complex systems, for which he has recently earned two best paper awards for his work in the mitigation of financial agent spoofing behaviors and an analysis of large language model use to manipulate other intelligent agents. His email address is d.byrd@bowdoin.edu and his Google Scholar page is at https://scholar.google.com/citations?user=lF8h840AAAAJ.