Towards a Programmable Framework for Agent Game Playing

Francis Lawlor
UCD School of Computer Science
Dublin, Ireland
francis.lawlor@ucdconnect.ie

Rem Collier
UCD School of Computer Science
Dublin, Ireland
rem.collier@ucd.ie

Vivek Nallur
UCD School of Computer Science
Dublin, Ireland
vivek.nallur@ucd.ie

ABSTRACT

The field of Game Theory provides a useful mechanism for modeling many decision-making scenarios. In participating in these scenarios individuals and groups adopt particular strategies, which generally perform with varying levels of success. However, most results have focussed on players that play the same game in an iterated fashion. This paper describes a framework which can be used to observe agents when they do not know in advance which game they are going to play. That is, the same group of agents could first play a few rounds of the Iterated Prisoner's Dilemma, and then a few rounds of the Linear Public Goods Game, and then a few rounds of Minority Game, or perhaps all games in a strictly alternating fashion or a randomized instantiation of games. This framework will allow for investigation of agents in more complex settings, when there is uncertainty about the future, and limited resources to store strategies.

KEYWORDS

Multi-Agent Systems, Game Theory, Self-Adaptation

1 INTRODUCTION

Turocy and von Stengel define Game Theory as "the formal study of decision-making where several players must make choices that potentially affect the interests of the other players" [33]. Game Theory provides the ability to reduce real world problems - those which are stylized in the form of a game - to mathematical models. This has resulted in the establishment of a huge array of formally recognized problems. Examples include the Minority Game [7], the Iterated Prisoners Dilemma [4], Public Goods Game [14], etc. However, most game theoretic approaches consider each game in isolation and do not investigate mechanisms, strategies or behaviours across games. This is inadequate for computational social simulations, since in reality human beings are confronted by many (possibly conflicting) games, and respond differently to each. Cognitive limitations on memory, time, resources, etc. combined with spatial influences result in actual behaviour that is not predicted/matched by rational agent simulations [32]. In parallel, the field of AI has been developing algorithms and techniques that focus on achieving human-level competence in several fields. Complex games with large state spaces such as Checkers [28] and Go [30], games with imperfect information such as Texas Hold'Em Poker [6] have been successfully played by algorithms. There are even algorithms that attempt to induce long-term cooperative behaviour in partly-competitive settings [9, 13]. However, to the best of our knowledge, there has been no attempt at finding a general bag of strategies that work in both competitive and cooperative settings, allow for the emergence of cooperation, altruism, and are flexible enough to be applied to many games. Human beings, on the other hand, have limited cognitive abilities and yet are able to (seemingly) effortlessly switch from domain to domain, and successfully compete as well as cooperate. We believe this is partly due to the inadequacy of computational simulation tools available for complex adaptive agents to play multiple games in heterogeneous environments. There are no tools, for example, that allow machine learning agents to be in competition with evolutionary agents across multiple heterogeneous games. This paper reports on a game-playing framework called *Arena*, that attempts to ameliorate the situation by providing an Agent-based reconfigurable environment for tournaments. A tournament, here, is defined to be multiple games played in some order, with players encountering each other *across* games. The primary goal of such a reconfigurable environment is that researchers can modify players, strategies, game rules, environments independent of each other, thus enabling not only a richer simulation but also a controlled increase in complexity. *Arena* aims to enable co-evolution of players, allowing the emergence of trust, reputation and coalitions as a natural consequence of known player identity across games.

2 RELATED WORK

Multi-Agent systems have been used for simulations of many problem domains such as Smart-grids [21], vehicular ad-hoc networks [11], smart buildings [36], e-procurement [35], cloud computing [23], healthcare [1], and transport [17]. However, many of these domains are complex enough that getting the *same* agent to adapt and perform in a cross-domain manner is very difficult. Desirable properties such as evolutionarily stable equilibria, allocative fairness etc. are also difficult to be formulated across multiple domains in an easily understandable manner. Game theory, on the other hand, has been used to mathematically compute or discover optimizing behaviour across a plethora of games, with a variety of constraints. However, it has not been used evaluate 'realistic' players that often encounter more than one game and suffer from resource constraints and bounded rationality. This paper attempts to create an intersection of game theory and multi-agent systems such that the advances in agent-modelling and simulation techniques can be applied in a well-understood games. Current approaches can roughly be divided into multi-agent simulations and game theory simulations.

Multi-Agent Simulations: Multi-Agent simulation environments are often general purpose environments, that focus

on providing ease of modelling of problem domain or agent behaviour or learning strategies. Most simulations tend to involve the creation of a bespoke implementation that is tied closely to the domain it is being evaluated against, with many implementations written in languages such as Netlogo [34] or environments such as Jason [5], Jade [2], etc. In spite of the sophistication of the agent environments, there are rarely any principled frameworks to test the **same** strategy/learning algorithm in **multiple** scenarios, since modelling multiple problem domains with any fidelity remains onerous. Hence, despite multiple reports of the importance and value of diversity for robustness in multi-agent systems [15, 31], to the best of our knowledge, there are no agent-based simulation platforms that allow for a systematic investigation of the same agent/strategy across heterogeneous problem domains.

Game Theory Simulations: Game theory abstracts away from the heterogeneity of problem domains and investigates stylized phenomena where the variables of interest are: payoffs, player moves and the presence of equilibria in conditions of repeated play. Most game theory simulators, to the best of our knowledge, focus on certain kinds of games. For instance, Gambit [18] is an attempt to build a generalized game playing framework for non-cooperative games where all players have access to a common set of strategies and the payoff for various moves is known at design time.

3 GENERALIZING GAME PLAYING

Current work of game playing tends to focus on the investigation of strategies for playing specific games, finding equilibria in repeated play or proving other properties for a specific game. However, games can vary across many dimensions, such as number of players (two-player, multi-player), moves (simultaneous, sequential), payoff (zero-sum, non-zero-sum), duration (repeated, one-shot), etc. In order to build a generalized game playing simulator, it is important to first agree on what constitutes a game and more generally a simulation. As was highlighted in the introduction, our system is specifically designed to support simulations that consist of multiple heterogeneous games played by agents using a diverse set of strategies. For example, a simulation may consist of 100 rounds of the Minority Game followed by 100 rounds of the Iterated Prisoners Dilemma. Further, the participants in this game may be broken down such that 30% are using a random strategy, 40% are using a Tit-for-Tat strategy, and 30% are using a random strategy. Interwoven into this model, we also envisage provision being made for periodic adaption/evolution steps whereby the participants are able to adjust/change their current strategy based on diverse means (ranging from random selection to a performance review or even the use of some form of genetic programming model).

Specifically, we envisage games to vary on the following axes:

- **Number of Players**: Two-Player, Multi-Player
- **Moves**: Simultaneous, Sequential
- **Payoff**: Zero-Sum, Non-Zero-sum, Rankings, Range of values (bounded and unbounded)

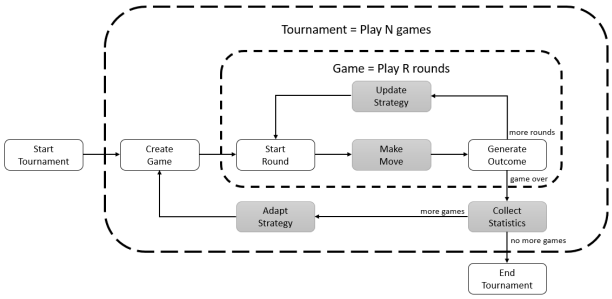

Figure 1: Stages of a Game

- **Identity**: Known, Unknown, Irrelevant
- **Communication Between Players**: Possible, Not possible
- **Topology**: Spatial, Non-spatial

Further, we envisage tournaments to vary on the following axes:

(1) **Communication Between Players**: Possible, Not possible
(2) **Game Order**: Ordered (known to players), Ordered (unknown to players), Random
(3) **Strategies**: Fixed-Bag-Fixed-Choice, Fixed-Bag-Random-Choice, Evolutionary Adaptation, Machine-learning adaptation

Note: These are not the only possible axes or options on the axes. Rather, these are the options that we are currently implementing. To represent these axes in a simulation, we have fixed on the following model. A simulation is a *tournament* that is played between a set of *agent* players who employ a fixed range of *strategies* (this can include meta-strategies that combine multiple sub-strategies). Each tournament consists of a number of *games* to be played, each of which consists of one or more *rounds*. Each round of a game involves an agent using their strategy to make a *move*. Each move is a bid / choice made by the agent that is relevant to the game being played. From here on, we refer to agents and players synonymously unless specifically required for reasons of clarity.

3.1 Decomposing Games

Based on our abstract model of a simulation, we propose, in figure 1, a generalized workflow for the execution of a tournament:

- *Start Tournament*: The tournament specification (what games, how many rounds, what agents, and what strategies) is loaded and the simulator initialized.
- *Create Game*: A game is created based on the corresponding specification and agents are linked to it.
- *Start Round*: Triggers the start of a round. This may involve transmission of a state to each agent or simply a request to make a move.

- *Make Move*: Here the agent must decide on its move which is based on the associated strategy. The agents move is submitted to the game.
- *Generate Outcome*: The game, once all required moves are made or a timer expires, decides on the outcome of the round (who wins and who loses, what is the payoff) and communicates this to the agents.
- *Update Strategy*: Here, the agent has the opportunity to update its strategy based on the outcome.
- *Collect Statistics*: When there are no more rounds in a game, the simulator gathers all specified statistics into a single resource that is stored for later analysis.
- *Adapt Strategy*: If there are more games to be played, then there is a chance for the agents to adapt their strategies. This may make use of some of the statistics gathered in the previous step. Adaptation can take any form that is appropriate, from parameter tuning to the use of evolutionary techniques, to machine learning techniques.
- *End Tournament*: The tournament finishes, statistics are collated into a data set that is released as a set of files and any requested summary statistics are presented to the user.

Of the above list of game stages, we believe that some are the responsibility of the simulator, some the game, and some by the strategy employed by the agent playing the game. For example, we feel that the stages in figure 1 that are shaded are the responsibility of the agent / strategy. While the others are the responsibility of the game / simulator. The delineation allows us to cleanly separate game design from strategy design which in turn allows us to develop generalized strategies that can be used to play a diverse range of games.

3.2 Generalized Game Design

To facilitate the integration of heterogeneous game types, we have attempted to create an abstract model of a game that can be customized as necessary.

Underpinning the model is the view of the player as an agent that interacts with artifacts within the game environment. These artifacts serve as enablers / constrainers of agent moves. Through the artifacts, each game specifies whether agents can recognize each other, whether there is an ordering to moves, how many rounds exist in a game, whether each round has an entry and exit condition, whether payoff / penalty is dealt to the agents after every round or it accumulates through the game. Thus, each game is envisioned as a configuration of artifacts. This allows new games to be created without having to re-write many common aspects of game playing. Figure 2 shows a sample code that is used to implement *both*, Iterated Prisoner's Dilemma as well as the Minority Game. This reconfiguration focused approach is extended deeper into agent-strategies as well, as shall be seen in the next sub-section. The artifact-based reconfiguration allows for another feature, evolutionary spread of features across agents. This means that agents that perform well can have their strategies copied and modified by other agents

```
rule $cartago.signal(string id,
          numberOfOptions(string agentId, int amountOfOptions)) {
    if (agentId == S.name())
        cartago.getGuess(S.name(), amountOfOptions);
}

rule $cartago.signal(string id,
        generatedGuess(string agentId, int guess, int numAgents)) {
    if (agentId == S.name()) {
        +numberOfAgents(numAgents);
        cartago.recordBid(agentId, guess, numAgents);
    }
}

rule $cartago.signal(string id,
        bidReceived(string agentId, int currentBid)) {
    if (agentId == S.name())
        cartago.playGame(S.name(), currentBid);
}

rule $cartago.signal(string id,
    gameFinished(string agentId, float payoff)) {
    if (agentId == S.name()) {
        query(numberOfAgents(int _numberOfAgents));
        cartago.recordPayoff(agentId, payoff, _numberOfAgents);
    }
}

rule $cartago.signal(string id, allPayoffsRecorded()) {
    cartago.receivedPayoff();
}

rule $cartago.signal(string id,
        sendUpdateKeyValuePair(string key, int value)) {
    cartago.updateStrategy(S.name(), key, value);
}
```

Figure 2: Interaction of Players with Cartago Artifacts

(via mechanisms such as clonal plasticity [24]), thus leading to evolutionary pressure on strategies.

3.3 Generalized Strategy Design

In order that strategies can be re-used across games, as well as new strategies added to a game, all strategies conform to an interface that agents can use to generate moves.

```
public interface IStrategy {
    public int generateChoice(HashMap<String, String> strategyResources);
    public void updateStrategy(String key, int value);
    public byte[] getIconAsBytes() throws IOException;
    public String getStrategyName();
    public String[] getAdditionalParameterNames();
}
```

Figure 3: Snippet of Strategy Interface

The *generateChoice* and *updateStrategy* methods are self-explanatory. The *strategyResources* map passed to *generateChoice* contains external data which may be required by the agent in decision-making. The parameters passed to *updateStrategy* are simply to be stored within the strategy object's internal resources map as a key-value pair. The remaining methods are required by the GUI for ensuring it reacts dynamically to user selection of strategies.

3.4 Worked out Example

The Iterated Prisoner's Dilemma (IPD) and the Minority Game (MG) are now compared to provide a concrete view of the generalization via artifacts. IPD and MG share three artifacts (see Table 1), while differing on three. Their implementation, therefore, is a simple composition of parametrized artifacts, which leads to faster and repeatable game creation.

	IPD	MG
Number of players	3 or more (always odd)	2
Moves	Simultaneous	Simultaneous
Amount of Payoff	Fixed	Fixed
Identity	Irrelevant	Known
Comm b/w players	Possible	Not Possible
Topology	Non-spatial	Non-spatial

Table 1: Comparing IPD and Minority Game

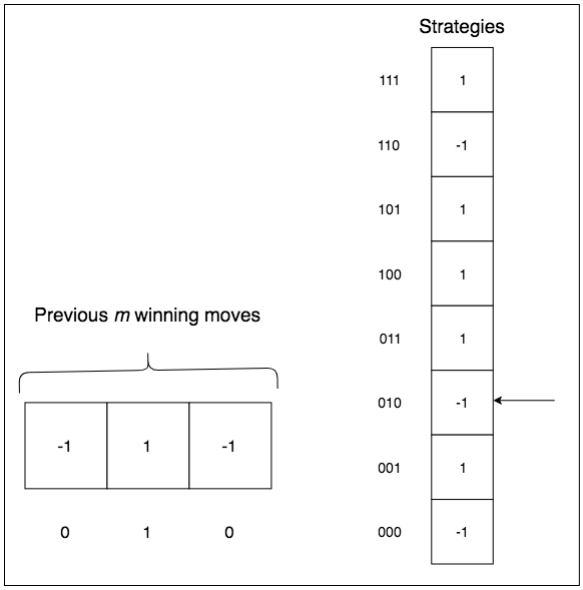

Figure 4: BestPlay example for m=3

Strategy Generalization: Generalizing strategies such that they can be re-used across games is a little more involved. Quite often, the strategy is very closely tied to the rules of the game or the number of available options to the agents. BestPlay [7] is a strategy created by the creators of the Minority Game. Recall that the MinorityGame (MG) consists of a population of `N` (odd) individuals, who have to make a simultaneous binary choice. The group that is in the minority, after making a choice, is the winner that receives a payoff. Although simple in its setup and play, the dynamics in MG has been used in multiple fields such as econophysics [3, 37], multi-agent resource allocation [16, 20], emergence of cooperation [10], and heterogeneity [12, 19]. The BestPlay strategy utilizes two pieces of information:**(a)** Memory of the winning moves from previous `m` rounds; **(b)** Vector containing a pool of strategies. The vector, for each player, is of length 2^m and is used to decide what to play next. Since MG allows only two moves, $1, -1$, the previous `m` winning moves can be encoded as a binary number (see Figure 4) The binary number can then be converted to an integer corresponding to a position in the strategy vector. In Figure 4, the strategy vector yields -1, as 010 corresponds to the integer 2, giving

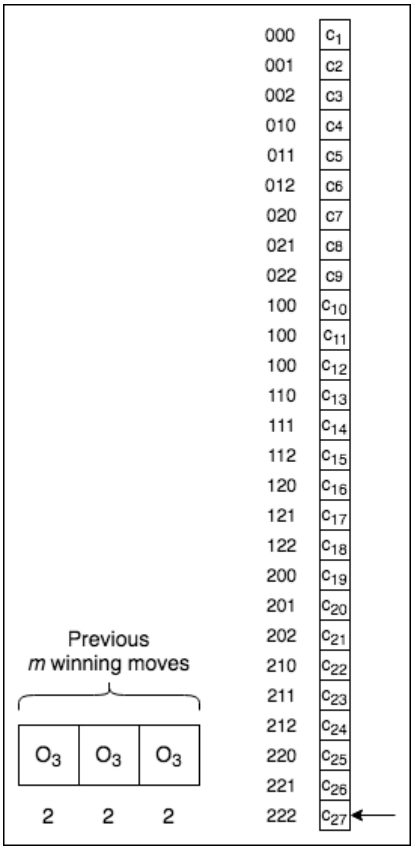

Figure 5: BestPlay example for m=3 and q=3

the result in the third position of the vector. Due to the nature of the encoding strategies, BestPlay is only applicable to games that involve selecting one out of two choices in every move. Since our intention is to allow the same strategy to be used across as many games as possible, we created a generalized implementation of BestPlay that retains its fundamental structure, but is not limited by two choices.

Generalizing BestPlay: The strategy vector can be generalized using a simple theorem that demonstrates how an `n`-ary code may be converted into an integer [22]. Given $a, b \in \mathbb{N}$ with $b>1$, there exist non-negative integers $x_0, x_1 ... x_n$ such that

$$a = x_0 + x_1 b + x_2 b^2 + \ldots + x_n b^n$$

with $0 \leq x_i < b$ and $x_n \neq 0$

Consider a game which presents three options (O_1, O_2, O_3) to a player. The strategy vector would now be q^m as opposed to 2^m, where q corresponds to the options available to a player. Continuing with the previous example of $m = 3$, the strategy vector now has a length of $3^3 = 27$. The choices (O_1, O_2, O_3) will be represented as options $0, 1, 2$. If the previous m winning moves were O_3, O_3, O_3, which corresponds to code 222, then BestPlay would index into the 26^{th} position of the strategy vector (see Figure 5).

This results in an implementation of the BestPlay strategy, as shown in Figure 6 and Figure 7, playable in games with arbitrary number of choices.

```
public ImplementedStrategy(HashMap<String, String> additionalParameters) {
    if (additionalParameters == null) {
        return;
    }

    this.numberOfChoices = Integer.parseInt(additionalParameters.get(ADDITIONAL_PARAMETER_NAMES[0]));
    this.choiceHistory = new ChoiceHistory(Integer.parseInt(additionalParameters.get(ADDITIONAL_PARAM
    this.strategyVectors.put(this.numberOfChoices, generateStrategyVector(choiceHistory.getChoiceHist
}

private int[] generateStrategyVector(int m, int _numberOfChoices) {
    int sizeOfStrategyVector = (int) Math.pow(_numberOfChoices, m);
    int[] strategyVector = new int[sizeOfStrategyVector];

    for (int i = 0; i < sizeOfStrategyVector; i++) {
        int randomChoice = (int) (Math.random() * _numberOfChoices);

        strategyVector[i] = randomChoice;
    }

    return strategyVector;
}
```

Figure 6: Constructor for BestPlay Strategy

```
public void updateStrategy(String key, int value) {
    if (key.equals(WINNING_CHOICE)) {
        this.choiceHistory.updateChoiceHistory(value);
    }
}
```

Figure 7: Generating a move using BestPlay Strategy

4 AN AOP-BASED GAME SIMULATOR

The goal of this work is to develop a flexible generalized game simulator through which we can study various properties of games and associated strategies. Specifically, we want to enable the development of a simulator that can support a heterogeneous set of games being played using a diverse suite of generalized strategies. We also wish to explore how these generalized strategies perform when applied to multiple game types that are played sequentially.

Typically, such simulators are implemented using languages such as NetLogo or using some form of generalized programming language (e.g. Java, C). In contrast, our approach is to explore the use of Agent-Oriented Programming (AOP) languages [29] and related technologies. Specifically we will use the ASTRA language [8], which is a variation of AgentSpeak(L) [26] together with CArtAgO [27] a framework that supports the modeling of the agents environment in terms of shared objects known as artifacts. This allows for modelling phenomena such as cultural learning where agents can copy strategies from successful neighbours.

A high level view of the proposed framework is highlighted in figure 8. In this figure, the stick people are agents, and the triangles are (CArtAgO) artifacts. In line with section 2, the game artifact specifies a generalized interface through which agents can interact with a game (e.g. make move) and the strategy artifacts are generalized artifacts that can be used to instantiate and use specific game strategies (e.g. random play, best play, ...). The register artifact provides a centralized list of players and their availability. When started, the *Tournament Master* reads the tournament specification and creates an initial community of *Player* agents (who

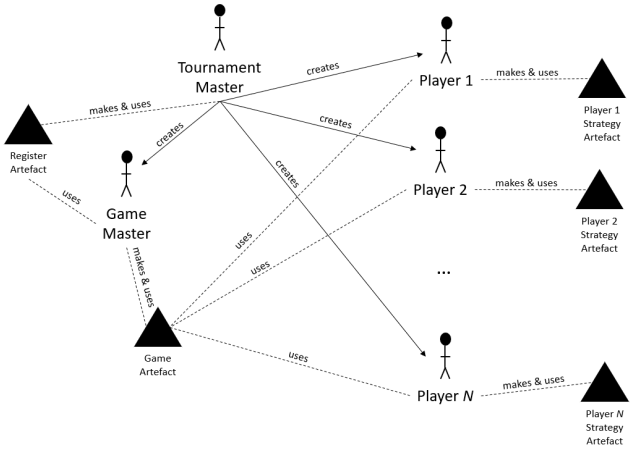

Figure 8: AOP-based Deployment

create associated strategy artifacts). These players are added to the registry. The *Tournament Master* then creates a *Game Master* who sets up a Game artifact that the *Player* agents connect to. The *Game Master* is responsible for the execution of the game. Its first task is to select a set of players to play the game. This is dependent on the selection policy adopted, and can be either a random selection of players of specific types or a fixed set of players. These players are invited to join the game, the game is played, and at the end, the *Game Master* informs the players that the game is over (causing the players to disconnect from the associated game artifact).

Our initial plan is to generate generic implementations of the *Tournament Master*, *Game Master*, and *Player* agents. These implementations will allow enough configuration to support their use with any game / basic game strategy. However, the motivation for using an AOP language to implement this is that, in the future (see our roadmap 5) we intend to start changing the *Player* agent to support more complex game playing behaviours. Due to the clear separation of concerns enforced through the use of an AOP language, we believe that this will be easier to achieve than if we had used a standard general purpose programming language. The current implementation can be accessed at: `https://gitlab.com/aop-arena`

5 ROADMAP

This paper describes a prototype, and is therefore only able to describe features that we have already implemented. However, we have a roadmap of features that we are working on implementing. Currently, all implementation is in the form of ASTRA agents and CArtAgO artifacts. However, this requires re-compilation of code whenever any aspect of the experiment changes. For easier experiment design and setup, we expect that more aspects need to be made into pure configuration text, so that non-coding-specialists can also use Arena. The features we expect to add to Arena are:

(1) **Open System with agents that enter and exit a tournament**: Complex domains such as urban modelling often require that simulations be 'open', *i.e.,* agents must be able to enter, exit and re-enter a simulation.
(2) **Game Description Language**: A human-readable and machine-parseable language for describing game rules, in terms of setup, number of players, sequential or simultaneous, competitive or cooperative or coalitional, kinds of payoffs, number of rounds, time-bound or not etc.
(3) **Player Description Language**: Players can diverge on the kinds of strategies that they implement, *i.e.,* they may be adaptive agents that adapt their strategies or machine learning agents that have *one* strategy that continuously adapts itself, or agents that possess a bag-of-strategies which they play in some order.
(4) **Tournament Description Language**: A human-readable and machine-parseable language for describing the order of games, number of repetitions for each game, open or closed system (can new agents enter in the middle of a tournament?)
(5) **Detection of Emergence within Tournament**: The presence of features such as known identity of players, multiple heterogeneous games, sophisticated strategies such as evolution / machine learning could lead to the emergence of 'agreements', unofficial rules, etc. which would be valuable to detect. We aim to incorporate tournament-wide emergence detection [25] mechanisms to allow for automated monitoring of large-scale tournaments.

6 CONCLUSIONS

This paper reports on a first prototype of a generalized game playing framework called *Arena*, that allows for the same agents to play multiple heterogeneous games. Arena currently allows strategies, game rules, and players to evolve independently of each other. As of this writing, we have implemented the Iterated Prisoner's Dilemma, Minority Game, Linear Public Goods Game, with agents that can play multiple rounds of each game, based on a tournament configuration. Strategies such as Tit-for-Tat, BestPlay have also been implemented in a generalized manner such that they can be re-used across games, by merely querying game parameters. We expect a more mature version of the framework to be a valuable tool, both for the agents community as well as game theory researchers.

REFERENCES

[1] Fabrice Axisa, P. M. Schmitt, C. Gehin, G. Delhomme, E. McAdams, and A. Dittmar. 2005. Flexible technologies and smart clothing for citizen medicine, home healthcare, and disease prevention. *IEEE Transactions on Information Technology in Biomedicine* 9, 3 (Sept 2005), 325–336. https://doi.org/10.1109/TITB.2005.854505

[2] Fabio Bellifemine, Giovanni Caire, and Dominic Greenwood. 2007. *Developing Multi-Agent Systems with JADE.* John Wiley & Sons, Ltd. https://doi.org/10.1002/9780470058411

[3] Ginestra Bianconi, Tobias Galla, Matteo Marsili, and Paolo Pin. 2009. Effects of Tobin taxes in minority game markets. *Journal of Economic Behavior & Organization* 70, 1-2 (May 2009), 231–240. http://www.sciencedirect.com/science/article/pii/S0167268108002084

[4] Ken Binmore. 2006. *The origins of fair play.* Report. Papers on economics and evolution.

[5] Rafael H. Bordini, Jomi Fred Hbner, and Michael Wooldridge. 2007. *Programming Multi-Agent Systems in AgentSpeak using Jason.* Wiley-Interscience. https://www.amazon.com/Programming-Multi-Agent-Systems-AgentSpeak-using/dp/0470029005?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0470029005

[6] M. Bowling, N. Burch, M. Johanson, and O. Tammelin. 2015. Heads-up limit holdem poker is solved. *Science* 347, 6218 (jan 2015), 145–149. https://doi.org/10.1126/science.1259433

[7] Damien Challet and Y-C Zhang. 1997. Emergence of cooperation and organization in an evolutionary game. *Physica A: Statistical Mechanics and its Applications* 246, 3-4 (1997), 407–418.

[8] Rem W. Collier, Seán Russell, and David Lillis. 2015. Reflecting on Agent Programming with AgentSpeak(L). In *PRIMA 2015: Principles and Practice of Multi-Agent Systems*, Qingliang Chen, Paolo Torroni, Serena Villata, Jane Hsu, and Andrea Omicini (Eds.). Springer International Publishing, Cham, 351–366.

[9] Jacob W. Crandall, Mayada Oudah, Tennom, Fatimah Ishowo-Oloko, Sherief Abdallah, Jean-François Bonnefon, Manuel Cebrian, Azim Shariff, Michael A. Goodrich, and Iyad Rahwan. 2018. Cooperating with machines. *Nature Communications* 9, 1 (jan 2018). https://doi.org/10.1038/s41467-017-02597-8

[10] Deepak Dhar, V. Sasidevan, and Bikas K. Chakrabarti. 2011. Emergent cooperation amongst competing agents in minority games. *Physica A: Statistical Mechanics and its Applications* 390, 20 (Oct. 2011), 3477–3485. https://doi.org/10.1016/j.physa.2011.05.014

[11] M. Ding, X. Cheng, and G. Xue. 2003. Aggregation tree construction in sensor networks. In *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, Vol. 4. 2168–2172 Vol.4. https://doi.org/10.1109/VETECF.2003.1285913

[12] Garrison W. Greenwood. 2009. Deceptive Strategies for the Evolutionary Minority Game. In *5th Intl Conf on Computational Intelligence and Games*. 25–31. http://dl.acm.org/citation.cfm?id=1719293.1719308

[13] Victoria Groom and Clifford Nass. 2007. Can robots be teammates?: Benchmarks in human–robot teams. *Interaction Studies* 8, 3 (oct 2007), 483–500. https://doi.org/10.1075/is.8.3.10gro

[14] R Mark Isaac, James M Walker, and Arlington W Williams. 1994. Group size and the voluntary provision of public goods: Experimental evidence utilizing large groups. *Journal of public Economics* 54, 1 (1994), 1–36.

[15] Peter R Lewis, Harry Goldingay, and Vivek Nallur. 2014. It's Good to Be Different: Diversity, Heterogeneity, and Dynamics in Collective Systems. In *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2014 IEEE Eighth International Conference on*. IEEE, 84–89.

[16] Yi Li and Robert Savit. 2004. Toward a theory of local resource competition: the minority game with private information. *Physica A: Statistical Mechanics and its Applications* 335, 1-2 (April 2004), 217–239. http://www.sciencedirect.com/science/article/pii/S0378437103011348

[17] A. S. Masoum, S. Deilami, P. S. Moses, M. A. S. Masoum, and A. Abu-Siada. 2011. Smart load management of plug-in electric vehicles in distribution and residential networks with charging stations for peak shaving and loss minimisation considering voltage regulation. *IET Generation, Transmission Distribution* 5, 8 (August 2011), 877–888. https://doi.org/10.1049/iet-gtd.2010.0574

[18] Richard D. McLennan McKelvey, Andrew M. Turocy, and L. Theodore. 2016. Gambit: Software Tools for Game Theory. Online. (2016). http://www.gambit-project.org/

[19] Bernardo A. Mello and Daniel O. Cajueiro. 2008. Minority games, diversity, cooperativity and the concept of intelligence. *Physica A: Statistical Mechanics and its Applications* 387, 2-3 (Jan. 2008), 557–566. http://www.sciencedirect.com/science/article/pii/S0378437107009995

[20] Richard Metzler and Christian Horn. 2003. Evolutionary minority games: the benefits of imitation. *Physica A: Statistical Mechanics and its Applications* 329, 3-4 (Nov. 2003), 484–498. https://doi.

org/10.1016/S0378-4371(03)00626-5

[21] A. H. Mohsenian-Rad, V. W. S. Wong, J. Jatskevich, R. Schober, and A. Leon-Garcia. 2010. Autonomous Demand-Side Management Based on Game-Theoretic Energy Consumption Scheduling for the Future Smart Grid. *IEEE Transactions on Smart Grid* 1, 3 (Dec 2010), 320–331. https://doi.org/10.1109/TSG.2010.2089069

[22] Victor H Moll. 2012. *Numbers and functions: from a classical-experimental mathematician's point of view*. Vol. 65. American Mathematical Soc.

[23] Vivek Nallur, Rami Bahsoon, Behzad Bordbar, and Xin Yao. 2010. Self-Adaptation of Web-Applications to Ensure Quality Attributes Using Market-Based Control. (2010).

[24] Vivek Nallur and Siobhán Clarke. 2018. Clonal plasticity: an autonomic mechanism for multi-agent systems to self-diversify. *Autonomous Agents and Multi-Agent Systems* 32, 2 (2018), 275–311. https://doi.org/10.1007/s10458-017-9380-x

[25] Eamonn O'Toole, Vivek Nallur, and Siobhan Clarke. 2017. Decentralised Detection of Emergence in Complex Adaptive Systems. *ACM Trans. Auton. Adapt. Syst.* 12, 1 (2017), 1–31. https://doi.org/10.1145/3019597

[26] Anand S Rao. 1996. AgentSpeak (L): BDI agents speak out in a logical computable language. In *European Workshop on Modelling Autonomous Agents in a Multi-Agent World*. Springer, 42–55.

[27] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. 2006. CArtAgO: A framework for prototyping artifact-based environments in MAS. In *International Workshop on Environments for Multi-Agent Systems*. Springer, 67–86.

[28] J. Schaeffer, N. Burch, Y. Bjornsson, A. Kishimoto, M. Muller, R. Lake, P. Lu, and S. Sutphen. 2007. Checkers Is Solved. *Science* 317, 5844 (jul 2007), 1518–1522. https://doi.org/10.1126/science.1144079

[29] Yoav Shoham. 1993. Agent-oriented programming. *Artificial intelligence* 60, 1 (1993), 51–92.

[30] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (jan 2016), 484–489. https://doi.org/10.1038/nature16961

[31] Hui Song, Amal Elgammal, Vivek Nallur, Franck Chauvel, Franck Fleurey, and Siobhán Clarke. 2015. On architectural diversity of dynamic adaptive systems. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, Vol. 2. IEEE, 595–598.

[32] Richard H. Thaler. 2016. *Misbehaving: The Making of Behavioral Economics*. W. W. Norton & Company. https://www.amazon.com/Misbehaving-Behavioral-Economics-Richard-Thaler/dp/039335279X?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=039335279X

[33] Theodore Turocy and Bernhard von Stengel. 2001. *Game Theory*. Technical Report CDAM Research Report LSE-CDAM-2001-09. London School of Economics. https://ia800302.us.archive.org/0/items/Cdam200109GameTheory/cdam-2001-09%20-%20game%20theory.pdf

[34] Uri Wilensky. 1999. Netlogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.. (1999). http://ccl.northwestern.edu/netlogo/

[35] F. Wu, G. Zsidisin, and A. Ross. 2007. Antecedents and Outcomes of E-Procurement Adoption: An Integrative Model. *IEEE Transactions on Engineering Management* 54, 3 (Aug 2007), 576–587. https://doi.org/10.1109/TEM.2007.900786

[36] J. H. Yoon, R. Baldick, and A. Novoselac. 2014. Dynamic Demand Response Controller Based on Real-Time Retail Price for Residential Buildings. *IEEE Transactions on Smart Grid* 5, 1 (Jan 2014), 121–129. https://doi.org/10.1109/TSG.2013.2264970

[37] Christopher A. Zapart. 2009. On entropy, financial markets and minority games. *Physica A: Statistical Mechanics and its Applications* 388, 7 (April 2009), 1157–1172. https://doi.org/10.1016/j.physa.2008.11.047