TAG: A Tabletop Games Framework

Raluca D. Gaina[*], Martin Balla[*], Alexander Dockhorn[*], Raúl Montoliu[†], Diego Perez-Liebana[*]
[*]School of Electronic Engineering and Computer Science, Queen Mary University of London, UK
[†]Insitute of New Imaging Technologies, Jaume I University, Castellon, Spain

Abstract

Tabletop games come in a variety of forms, including board games, card games, and dice games. In recent years, their complexity has considerably increased, with many components, rules that change dynamically through the game, diverse player roles, and a series of control parameters that influence a game's balance. As such, they also encompass novel and intricate challenges for Artificial Intelligence methods, yet research largely focuses on classical board games such as *chess* and *Go*. We introduce in this work the Tabletop Games (TAG) framework, which promotes research into general AI in modern tabletop games, facilitating the implementation of new games and AI players, while providing analytics to capture the complexities of the challenges proposed. We include preliminary results with sample AI players, showing some moderate success, with plenty of room for improvement, and discuss further developments and new research directions.

1 Introduction

In the last few decades, tabletop games have gone through a 'Renaissance', gaining more popularity than ever: thousands of them are published each year and welcomed by an expanding audience of gamers (Engelstein and Shalev 2019). Many of these games are typically designed with richer mechanics and rules, and less focus on chance-based elements. Modern tabletop games are very diverse and complex in general, which can provide various challenges, such as unique game state representations, partial observability on various levels, actions outside of a player's turn, cooperation, and competition in the same game, etc. Therefore, they have a very different and complex set of mechanisms that would require a lot of effort to develop using previous approaches.

AI Research in board games has mostly been, with some exceptions, focused on traditional board games, either in isolation (*chess*, *Othello*, *Go*, etc.) or as part of general game playing (GGP) frameworks, such as GGP (Genesereth, Love, and Pell 2005), Ludii (Piette et al. 2019) and OpenSpiel (Lanctot and others 2019). While these frameworks also allow the definition of additional games, they are limited to common mechanics or require extensive development effort.

In this work, we introduce the Tabletop Games (TAG) framework, which is a collection of tabletop games, agents, and development tools meant to provide a common platform for AI research. This work is mainly motivated by three factors: i) the characteristics of modern tabletop games (multi-player, partially observable, large action spaces, competition, and collaboration, etc.) provide an interesting challenge to AI research. These modern games provide many characteristics not implemented in existing frameworks, e.g. changing player roles or varying forms of cooperative and competitive game-play; ii) TAG presents tabletop games from a GGP perspective, by providing a common API for games and playing agents; and iii) we aim to provide a framework that can incorporate different games under a common platform, making it possible for the research community to implement their own games and AI players to expand TAG's collection. De Araujo et al. (2019) highlighted the need for such a framework in a survey that described the different schemes and data structures used in the literature of digital board games.

In order to provide the necessary flexibility to support the great variety of existing tabletop games, TAG requires the user to implement games via a programming language (Java), instead of using a game description language as other general frameworks do. TAG provides many customisable components to simplify the development of additional games not currently in the framework. It is able to handle partial observability, providing simple means of adding custom observation schemes, game-state heuristics, and agent statistics, as well as supporting the development of graphical user interfaces for human play with computer players.

While we are working on extending the framework with additional game mechanics, games and agents, the current version is publicly available[1]. The contributions of this paper are two-fold: firstly, we present the framework, its structure, games and AI players implemented (see Section 3). Secondly, we provide a discussion on a baseline experimentation (Section 4), aimed at illustrating insights into the games implemented, their features, and performance of vanilla AI players. In Section 5 we discuss challenges and opportunities included for this framework and we conclude with final remarks on future developments in Section 6.

[1]https://github.com/GAIGResearch/TabletopGames

2 Related Work

AI research and board games have been closely related since the beginnings of the field, with game-playing agents for *Tic-Tac-Toe*, *Checkers* and *chess* (Campbell and others 2002), and the more recent breakthroughs in *Go* (Silver et al. 2017). One of the most well-known contests, the General Game Playing (GGP) competition (Genesereth, Love, and Pell 2005), featured classical board games written in the Game Description Language and promoted research into generic game-players. (Piette et al. 2019) later introduced the "ludemic" general game system Ludii, which builds upon ideas from GGP. Ludii defines games as structures of *ludemes*, high-level, easily understandable game concepts, which allows for concise and human-understandable game descriptions, as well as easy implementation of new games. The current main focus of the *Ludii* project is on classical and ancient board games. Kowalski et al. presented in (Kowalski et al. 2019) a new GGP language, called Regular Boardgames (RBG), with a similar focus as Ludii, but which describes games as regular expressions instead, for increased efficiency.

We consider direct code implementations to be more accessible, faster to execute and easier to work with in many cases. More similarly to our framework in this regard, *OpenSpiel* (Lanctot and others 2019) provides a collection of games and algorithms written in C++ and exposed to Python. Most of their games are still traditional board games, with some exceptions, such as the inclusion of *Hanabi*. Differently, TAG shifts the development effort onto the framework, rather than the games, by making a wide range of components, rules, actions, etc. available to users. Our system allows for fast prototyping of new games and immediate extraction of insights and features of interest in each game through readily-available game and AI-facilitated analysis. We further support many research directions, from simulation-based game-playing to parameter optimisation of games and artificial players.

Here, we focus on more types of games, including board, card, dice and role-playing games, among others, often grouped under the *tabletop games* umbrella term. *Tabletop Simulator* (Henry 2015) is an example of software facilitating implementation of tabletop games components in a physics-based simulated environment; this allows players to interact with the games as they would in the real world, but the many games implemented lack support for automatic rule execution, nor does the software facilitate AI research as targeted with TAG. However, tabletop games research has been gaining popularity in recent years. Research in game-playing agents for card games is common in competitive (*Poker* (Moravčík and others 2017) and *Bridge* (Cazenave and Ventos 2019)) and cooperative (*Hannabi* (Bard and others 2020)) games.

Asymmetric player roles is one feature often encountered in modern tabletop games, and these have been studied in games such as *The Resistance* (Serrino et al. 2019) and *Ultimate Werewolf* (Eger and Martens 2019). This is just another complexity added in modern tabletop games, yet more lead to the need for intricate strategic planning. To this extent, Monte Carlo Tree Search (MCTS) methods have been tried in *Settlers of Catan* (Szita, Chaslot, and Spronck 2009) and *Risk* (Gibson, Desai, and Zhao 2010), and Rolling Horizon Evolutionary Algorithms (RHEA) in *Splendor* (Bravi et al. 2019), all showing a great improvement in performance. Other games have been more recently highlighted as important challenges for AI players due to their strategic complexity (*Pandemic* (Chacón and Eger 2019; Sfikas and Liapis 2020)) or very large action spaces (*Blood bowl* (Justesen et al. 2019)).

Research has not only focused on game-playing AI, however. *Ticket to Ride* (de Mesentier Silva et al. 2017) was used as an example for employing AI players for play-testing games, characterising their features based on different play-styles and finding possible bugs or gaps in the ruleset. Further, the use of Procedural Content Generation for such games is highlighted by (Guzdial et al. 2020); given the rule complexities and the multitude of components in modern tabletop games, AI methods can provide a more efficient way of searching the possibility space for interesting variations.

The Tabletop Games (TAG) framework introduced in this paper brings together all of these different research directions and provides a common ground for the use of AI algorithms in a variety of tabletop games, removing the effort of creating different frameworks for different purposes and simplifying the overall development process. As far as we know, TAG is the first framework that allows the development of multiple games and AI players under a common API for complex modern tabletop games.

3 The Framework

TAG was designed to capture most of the complexity that modern tabletop games provide, with a few games implemented already and more in progress.

3.1 Concepts

Our framework includes handy definitions for various concepts and components common across many tabletop games (Engelstein and Shalev 2019).

We define an **action** as an independent unit of game logic that modifies a given game state towards a specific effect (e.g. player draws a card; player moves their pawn). These actions are executed by the game players and are subject to certain **rules**: units of game logic, part of a hierarchical structure (a game flow graph). Rules dictate how a given game state is modified and control the flow through the game graph (for instance, checking the end of game conditions and the turn order). This **turn order** defines which player is due to play at each time, possibly handling player reactions forced by actions or rules. At a higher level, games can be structured in **phases**, which are time frames where specific rules apply and/or different actions are available for the players.

All tabletop games use **components** (game objects sharing certain properties), whose state is modified by actions and rules during the game. TAG includes several predefined components to ease the development of new games, such as tokens (a game piece of a particular type), dice (with N sides), cards (with text, images or numbers), counters (with a numerical value), grid and graph boards. Components can also be grouped into collections: an **area** groups components

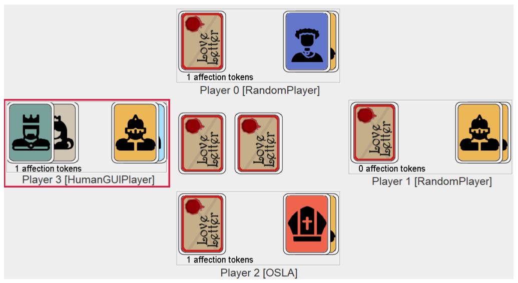

Figure 1: GUI for *Love Letter*, red line shows current player.

in a map structure in order to provide access to them using their unique IDs, while a **deck** is an ordered collection with specific interactions available (e.g. shuffle, draw, etc.). Both areas and decks are considered components themselves.

3.2 Structure

The TAG framework brings together all of the concepts and components described previously and allows quick implementation and prototyping of new games. To this end, a flexible API is provided for all functionality needed to define a game, with multiple abstract classes that can be extended for specific implementations. The framework provides some generic functionality, from ready-made components, rules, actions, turn orders and game phases, to a fully functional game loop and a prototyping GUI. The GUI allows users to start interacting with the games as soon as they have the two main classes required set up: a *Game State* (GS) class, and a *Forward Model* (FM) class.

GS is a container class, including all variables and game components which would allow one to describe one specific moment in time. It defines access methods in the game state to retrieve existing game components, make custom and partially observable copies of the state, and define an evaluation function that can be used by the playing agents. The **FM** encompasses the logic of the game: performs the game setup, defines what actions players can take in a particular game state, applies the effect of player actions and any other game rules applicable, uses a turn order to decide which player is due to play next (or may wait for all players to return an action before processing for simultaneous-actions games), and checks for any end of game conditions. The FM is available to AI players for game simulations.

For each game, users can further implement specific actions, rules, turn orders, game parameters (for easy modification of game mechanics), a GUI and provision of game data. The last is useful when the game requires large amounts of data such as tile patterns, cards and board node connections, and it is provided via JSON files. A full guide on using the framework and implementing new games is available in the wiki provided with the code and in (Gaina et al. 2020).

3.3 Games

There are currently 7 games implemented in the framework, varying from very simple test games (*Tic-Tac-Toe*) to strategy games (*Pandemic* (Leacock 2008)), as well as diverse challenges for AI players. A few games are currently in active development (*Descent* (Fantasy Flight Publishing, Inc. 2012), *Carcassonne* (Wrede 2000) and *Settlers of Catan* (Teuber 1995)), and many more are in the project's backlog, including games from other frameworks to allow for easy comparison (see Section 2).

All games implemented can be found in the `games` package, each registered in the `games.GameType` class; this class allows specifying properties for each game, to allow for automatic listing for experiments (e.g. a list of all games with the "cooperative" tag). We highlight next some particularities of the games currently implemented in the framework.

Tic-Tac-Toe 2 players alternate placing their symbol in a $N \times N$ grid until one player completes a line, column or diagonal and wins the game; if all cells in the grid get filled up without a winner, the game is a draw.

This is the simplest game included in the framework, meant to be used as a quick reference for the minimum requirements to get a game up and running. Its implementation makes use of mostly default concepts and components, but it implements a scoring heuristic and a custom GUI for an easier interaction given the specific game mechanics.

Love Letter (Kanai 2012) 2 to 4 players start the game with one card each, representing a character, a value and a unique effect. A second card is drawn at the start of each turn, one of which must be played afterwards. After the last card of the deck is drawn, the player with the highest valued card wins the current round. A player wins the game after winning 5 rounds. *Love Letter* features partial observability, asymmetric and changing player roles and a point system over several rounds. Figure 1 shows an example game state.

Uno (Robbins 1971) The game consists of coloured cards with actions or numbers. Numbered cards can only be played in case either the colour or the number matches the newest card on the discard pile. Action cards let 2 to 10 players draw additional cards, choose the next colour to be played or reverse the turn order. A player wins after gaining a number of points over several rounds (computed as the sum of all other players' card values). *Uno* features stochasticity, partial observability and a dynamically changing turn order.

Virus! (Cabrero and others 2015) 2 to 6 players have a body each that consists of four organs, which can be: infected (by an opponent playing a virus card), vaccinated (by a medicine card), immunised (by 2 medicine cards) or destroyed (by opponents playing 2 consecutive virus cards). The winner is the first player who forms a healthy and complete body. *Virus!* features stochasticity and partial observability, with the draw pile and opponents' cards being hidden.

Exploding Kittens (Inman and others 2015) 2 to 5 players try to avoid drawing an exploding kitten card while collecting other useful cards. Each card gives a player access to unique actions to modify the game-state, e.g. selecting the player taking a turn next and shuffling the deck. This game features stochasticity, partial observability and a dynamic turn order with out-of-turn actions: in contrast to previous games, *Exploding Kittens* keeps an action stack so that players have the chance to react to cards played.

Colt Express (Raimbault 2014) 2 to 6 players control a bandit each, with a unique special ability. Their goal is to collect the most money while traversing the two-level compartments in a train and avoiding the sheriff (a non-playing character moved by players and round card events). The game consists of several rounds, each with a planning (players play action cards) and an execution (cards are executed in the same order) phase. This processing scheme forces players to adapt their strategy according to all the moves already played, in an interesting case of partial observability and non-determinism: the opponents' type of action may be known (sometimes completely hidden in a round), but not how it will be executed. Additionally, the overall strategy should be adapted to a bandit's unique abilities.

Pandemic (Leacock 2008) *Pandemic* is a cooperative board game for 2 to 4 players. The board represents a world map, with major cities connected by a graph. Four diseases break out and the objective of the players is to cure them all. Diseases keep spreading after each player's turn, sometimes leading to outbreaks. Each player is assigned a unique role with special abilities and is given cards that can be used for travelling between cities, building research stations or curing diseases. Additionally, they have access to special event cards, which can be played anytime (also out-of-turn). All players lose if they run out of cards in the draw deck, if too many outbreaks occur or if the diseases spread too much. *Pandemic* features partial observability with face-down decks of cards and asymmetric player roles. It employs a reaction system to handle event cards and is the only game currently using the graph-based rule system.

3.4 AI Players

All implemented players follow a simple interface, only requiring one method to be implemented: `getAction`. This receives a game state object reduced to the specific player's observation of the current state of the game. How this reduced game state is built is game-dependent, usually randomising unknown information. This method expects an action to be returned out of those available and is called whenever it is the player's turn and they have more than 1 action available (i.e. the player actually has a decision to make). If no decision is required, the agent can choose to still receive and process the information on the game state (in the `registerUpdatedObservation` function) but an action is not requested. They may also choose to implement the `initializePlayer` and `finalizePlayer` functions which are called at the beginning and end of the game, respectively. Each player has a player ID assigned by the game engine, and they receive the forward model of the game currently being played. The FM can then be used to advance game states given actions, compute actions available, or reset a game to its initial state. The rest of this section defines the sample players implemented in the framework. These agents use the game's score to evaluate game states (as implemented on the game side), but their heuristic functions may be swapped with a different object implementing the `IStateHeuristic` interface. Custom heuristics take the current state as input and return a scalar number representing the value of that state without any other restrictions. Agents can be given a heuristic function on initialisation and then instead of using the reward directly from the game they process every state they visit using the provided heuristic.

Human Players Two types of human interaction are available, both of which interrupt the game loop to wait for human actions on their turn. **Console** allows human play using the console. It outputs the game state and available actions in the console and the player inputs the index of the action they choose to play. **GUI** allows human play with a Graphical User Interface, which is game-specific. It uses an `ActionController` object to register player action requests, which are then executed in the game.

Random The simplest automatic player chooses random actions out of those available on its turn.

One Step Look Ahead (OSLA) A greedy exhaustive search algorithm, it evaluates all actions from a given game state and picks that which leads to the highest valued state.

Rolling Horizon Evolutionary Algorithm (RHEA) RHEA (Perez-Liebana et al. 2013) evolves a sequence of $L = 10$ actions over several generations, choosing the first action of the best sequence found to play in the game. The algorithm is randomly initialised with a sequence of actions. At each generation it creates a mutation of the current best solution, keeping the best solution of the two. This process repeats until the given budget is exhausted.

Given the variable action spaces and that actions available are highly dependent on the current game state, the mutation operator chooses a gene in the individual (i.e. position in the action sequence) and changes all actions from that point until the end of the individual to new random valid actions. The game's forward model is therefore used in both mutation (to advance the game state given the last action, in order to find the available actions for the given position in the sequence) and evaluation (game states reached through the sequence of actions are evaluated using the game's heuristic, added up for a discounted total with discount factor $\gamma = 0.9$, and this total becomes the fitness of the individual). It is important to note that RHEA evolves only its own actions and opponents are given a random model (with intermediate states after opponent actions ignored in fitness evaluations).

Monte Carlo Tree Search (MCTS) MCTS (Browne and others 2012) incrementally builds an asymmetric game tree balanced towards to most promising parts of the game state space. It uses multiple iterations of four steps: first, it navigates through the tree, using a *tree policy*, until reaching a node which is not yet fully expanded; next, it adds a new random child of this node to the tree; it then performs a Monte Carlo rollout from the new child (randomly sampling actions until the end of the game or a predetermined depth $L = 10$); the state reached at the end of the rollout is evaluated with a heuristic function, and this score is backed up through all the nodes visited during the iteration. The process is repeated

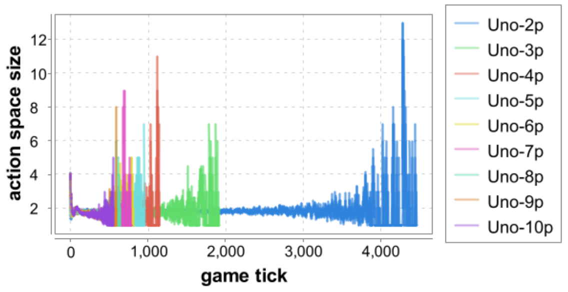

Figure 2: Action space size in *Uno* with all player number versions; 1000 runs per version played by random players.

until the budget is exhausted, and the most visited child of the root is chosen as the action to play.

The version implemented in the framework is closed-loop: it stores game states in each of the nodes. Further, the rollout step was removed after initial experiments showing an increased performance without it; therefore, the forward model of the game is only used when expanding a leaf node. The resulting node is immediately evaluated using the heuristic and its value is backed up through the tree.

4 Discussion

This section gives an overview of game analytics which can be extracted from all games, as well as some preliminary results for the sample AI players described in Section 3.4.

4.1 Game Analysis

All games in the framework can be analysed to illustrate the challenge they provide for AI players, with the following metrics currently readily available:

Action space size: the number of actions available for a player on their turn (e.g. Figure 2). **Branching factor**: the number of distinct game states reached through player actions from a given state. **State size**: the number of components in a state. **Hidden information**: the percentage of components hidden from players on their turn. **Game speed**: the execution speed of 4 key functions (in number of calls per second): setup, next, available action computation and state copy. **Game length**: measured as the number of decisions taken by AI players, the total number of game loop iterations (or ticks), the number of rounds in the game and the number of actions per turn for a player. **Reward sparsity**: granularity of the heuristic functions provided by the game, measured by min, max and standard deviation of rewards seen by players.

When looking at the games currently implemented, the first thing to note is that all games are very fast to execute: most games can execute over 1 million calls per second to the (usually) most expensive functions (`next` and `copy`). The games vary in length, with only 7.61 ticks for the simplest game, *Tic-Tac-Toe*, but 540.78 for *Uno*. We further see variations in the state size, with *Pandemic* showing most complex, while *Uno* includes the most hidden information. *Love Letter* shows its strategic complexity through the higher branching factor (10.78), while *Exploding Kittens* boasts one of the largest spread of rewards.

4.2 Baseline AI Player Performance

We tested the performance of the sample agents on each of the implemented games. For *Tic-Tac-Toe*, we report the win-rate per agent when playing 100 times against every possible opponent. Since *Pandemic* is a cooperative game, we report results from games played with a team of 4 instances of the same agent, e.g. 4 MCTS players. For the remaining games, we report the average win rate per agent when playing in a 4-player match against one instance of all other agents. All but the random agent are using state evaluation functions that are provided with each game. For both search-based algorithms, we use a budget of 4000 calls to the `FM.next()` function.

The average win-rate per games is shown in Table 1. Our results indicate the MCTS agent to be the best, achieving the highest average win rate in 5 out of 6 competitive games, with an overall win rate of 45.5% in these games - thus clearly dominating the other agents. While RHEA also outperforms random in most of the games (with the exception of *Uno*), it still falls behind the OSLA agent. This could be due to the large uncertainty built up in its rigid sequences of actions (as opposed to the flexible game trees built by MCTS) in these games with partial observability and stochasticity, where a greedy approach appears to be preferable. Further, some games tested are simple enough that a greedy approach works best and are highly advantaged by the heuristics provided by each game (e.g. *Tic-Tac-Toe*).

Additionally, we note that no agent is able to win in the cooperative game *Pandemic*, as they are unable to perform the multi-faceted long-term planning required to avoid all of the loose conditions and win the game. Further analysis, such as the distance from winning game states for each AI team, could show interesting insights into these agents' capabilities. Moreover, we observe *Uno* and *Colt Express* as cases where the performance between all players, including random, is very close ($22-26\%$ in *Uno* and $19-29\%$ in *Colt Express*). This highlights the difficulty of the types of problems proposed, as well as the importance of the heuristic chosen for a game, as some features of a game state may prove deceiving.

However, the statistical forward planning methods described here (MCTS and RHEA) benefit from ample literature with a large parameter space each, which could be tuned for increased performance.

5 Challenges and Opportunities

The presented framework opens up several directions of research and proposes a variety of challenges for AI players, be it search/planning or learning algorithms. Its main focus is to promote research into General Game AI that is able to play many tabletop games at, or surpassing, human level. Relatedly, the agents should be able to handle both **competitive** (most common testbeds in literature), **cooperative** and even **mixed** games. For instance, a future planned development is the inclusion of the game *Betrayal at House on the Hill* (Glassco and others 2004), in which the players start off playing cooperatively to later split into teams mid-way through the game, from which point on they are competing instead with newly given team win conditions and rules. Most tabletop games include some degree of **hidden**

Table 1: AI player performance, 100 game repetitions. Highest win rate in bold. *Tic-Tac-Toe* played in a round-robin tournament. *Pandemic* uses 4 instances of the same agent. All others played in their 4-player variants, with 1 instance of each agent.

	Tic-Tac-Toe	Love Letter	Uno	Virus!	Expl. Kittens	Colt Express	Pandemic	Total
Random	0.12	0.00	**0.26**	0.01	0.05	0.19	0.00	0.09
OSLA	0.45	0.24	**0.26**	0.27	**0.37**	**0.29**	0.00	0.27
RHEA	0.44	0.32	0.22	0.3	0.21	0.28	0.00	0.25
MCTS	**0.98**	**0.44**	**0.26**	**0.42**	**0.37**	0.26	0.00	**0.39**

information (e.g. face-down decks of cards) and many more players compared to traditional video-game AI testbeds, introducing higher levels of uncertainty. However, such games often make use of similar mechanics, even if in different forms: thus **knowledge transfer** would be a fruitful area to explore, so that AI players can pick up new game rules more easily based on previous experiences, similar to how humans approach the problem. Some tabletop games further feature **changing rules** (e.g. *Fluxx* (Looney and Looney 1997)) which would require highly adaptive AI players, able to handle changes in the game engine itself, not only the game state. Many others rely on large amounts of content and components, for which the process of creating new content or modifying the current one for balance, improved synergies etc. could be improved with the help of Procedural Content Generation methods (e.g. cards for the game *Magic the Gathering* (Garfield 1993) were previously generated in a mixed-initiative method by (Summerville and Mateas 2016)).

Specific types of games can also be targeted by research, an option highlighted by TAG's categorisation and labelling of games and their mechanics. Thus AI players could learn to specialise in games using certain mechanics or in areas not yet explored, such as **Role-Playing** or **Campaign** games (i.e. games played over several linked and progressive sessions). These games often feature **asymmetric player roles**, with a special one highlighted (the dungeon master) whose aim is to control the enemies in the game in order to not necessarily win, but give the rest of the players the best experience possible and the right level of challenge. Strategy AI research could see important applications in this domain, as many tabletop games include elements of strategic planning. Role-playing games focused more on the story created by players (e.g. *Dungeons and Dragons*) rather than combat mechanics (e.g. *Gloomhaven*) would also be a very engaging and difficult to approach topic for AI players, where Natural Language Processing research could take an important role.

The framework enables research into **parameter optimisation**: all parameter classes for games, AI players or heuristics can implement the `ITunableParameters` interface; parameters can then be automatically randomised, or more intelligently tuned by any optimisation algorithm. This allows for quick and easy exploration of various instances of a problem, a potential increase in AI player performance, or adaptation of AI player behaviour to user preference.

We have mentioned previously that the games implemented offer reduced observations of the game state to the AI players, based on what they can currently observe. These hidden information states (usually) do not keep a history of what was previously revealed to a player. Instead, the AI players should learn to memorise relevant information and build **belief systems**, as humans would in a real-world context - a very interesting direction of research encouraged by TAG.

Lastly, the framework includes the possibility for games to define their states in terms of either **vector observations** (`IVectorObservation`), which enables learning algorithms to be easily integrated with the framework; or **feature-based observations** (`IFeatureRepresentation`), which allows for more complex algorithms which can perform a search in the feature space of a game, rather than the usual game state space approached.

6 Conclusion

This paper introduces the new Tabletop Games (TAG) framework, which aims to promote research into general Artificial Intelligence with features for easy implementation and bridge-building between tabletop games and artificial players, with some examples already included. We further analyse the games in the framework, showing a wide variety of action spaces, information available to the AI agents, duration etc., as well as tasks and challenges introduced. The AI player performance analysis shows Monte Carlo Tree Search to dominate all other sample agents in the framework, with simple greedy methods being surprisingly competitive in some of the games. Overall, however, the problems proposed are far from being solved.

The framework opens up and facilitates many directions of research, and yet many more developments are possible and planned. More measurements for both games and AI players can be added, to paint a more complete picture of the challenges the players would face, as well as the current state of available methods for such games: skill depth, overall state space size, stochasticity, size of search trees, player role asymmetry - all would give a much more in-depth view of the framework as a whole, aiding in future developments of both tabletop games and general AI players.

Further, we aim to facilitate interfacing external games with our framework in order to gain the full benefits of the in-depth analysis and interaction with the implemented players without the need to re-implement everything from scratch: this would open up the framework to many more already existing games, and also increase the number and complexity of environments the AI players can be exposed to, improving their quality as well.

Acknowledgments

This work was partly funded by the EPSRC CDT in Intelligent Games and Game Intelligence (IGGI) EP/L015846/1 and EPSRC research grant EP/T008962/1.

References

Bard, N., et al. 2020. The Hanabi Challenge: A New Frontier for AI Research. *Artificial Intelligence* 280:103216.

Bravi, I.; Perez-Liebana, D.; Lucas, S. M.; and Liu, J. 2019. Rinascimento: Optimising Statistical Forward Planning Agents for Playing Splendor. In *2019 IEEE Conference on Games (CoG)*, 1–8. IEEE.

Browne, C. B., et al. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in games* 4(1):1–43.

Cabrero, D., et al. 2015. *Virus!* El Dragón Azul.

Campbell, M., et al. 2002. Deep Blue. *Artificial Intelligence* 134(1-2):57–83.

Cazenave, T., and Ventos, V. 2019. The alpha-mu Search Algorithm for the Game of Bridge. *arXiv preprint arXiv:1911.07960*.

Chacón, P. S., and Eger, M. 2019. Pandemic as a Challenge for Human-AI Cooperation. In *Proceedings of the AIIDE workshop on Experimental AI in Games*.

de Araujo, L. J. P.; Charikova, M.; Sales, J. E.; Smirnov, V.; and Thapaliya, A. 2019. Towards a Game-Independent Model and Data-Structures in Digital Board Games: an Overview of the State-of-the-Art. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, 1–8.

de Mesentier Silva, F.; Lee, S.; Togelius, J.; and Nealen, A. 2017. AI-Based Playtesting of Contemporary Board Games. In *Proceedings of the International Conference on the Foundations of Digital Games - FDG'17*. ACM Press.

Eger, M., and Martens, C. 2019. A Study of AI Agent Commitment in One Night Ultimate Werewolf with Human Players. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 15, 139–145.

Engelstein, G., and Shalev, I. 2019. *Building Blocks of Tabletop Game Design: An Encyclopedia of Mechanisms*. CRC Press LLC.

Fantasy Flight Publishing, Inc. 2012. *Descent: Journeys in the Dark 2nd Edition*. Diamond Comic Distributors.

Gaina, R. D.; Balla, M.; Dockhorn, A.; Montoliu, R.; and Perez-Liebana, D. 2020. Design and Implementation of TAG: a Tabletop Games Framework. *arXiv preprint arXiv:2009.12065*.

Garfield, R. 1993. *Magic: The Gathering*. Wizards of the Coast.

Genesereth, M.; Love, N.; and Pell, B. 2005. General Game Playing: Overview of the AAAI Competition. *AI magazine* 26(2):62–62.

Gibson, R.; Desai, N.; and Zhao, R. 2010. An automated technique for drafting territories in the board game Risk. In *Sixth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Glassco, B., et al. 2004. *Betrayal at House on the Hill*. Avalon Hill Games, Inc.

Guzdial, M.; Acharya, D.; Kreminski, M.; Cook, M.; Eladhari, M.; Liapis, A.; and Sullivan, A. 2020. Tabletop Roleplaying Games as Procedural Content Generators.

Henry, J. 2015. *Tabletop Simulator*. Berserk Games.

Inman, M., et al. 2015. *Exploding Kittens*. Ad Magic, Inc.

Justesen, N.; Uth, L. M.; Jakobsen, C.; Moore, P. D.; Togelius, J.; and Risi, S. 2019. Blood Bowl: A New Board Game Challenge And Competition For AI. In *2019 IEEE Conference on Games (CoG)*, 1–8. IEEE.

Kanai, S. 2012. *Love Letter*. Alderac Entertainment Group.

Kowalski, J.; Mika, M.; Sutowicz, J.; and Szykuła, M. 2019. Regular Boardgames. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 1699–1706.

Lanctot, M., et al. 2019. OpenSpiel: A Framework for Reinforcement Learning in Games. *CoRR* abs/1908.09453.

Leacock, M. 2008. *Pandemic*. Z-Man Games, Inc.

Looney, A., and Looney, K. 1997. *Fluxx*. Looney Labs.

Moravčík, M., et al. 2017. Deepstack: Expert-Level Artificial Intelligence in Heads-Up No-Limit Poker. *Science* 356(6337):508–513.

Perez-Liebana, D.; Samothrakis, S.; Lucas, S. M.; and Rolfshagen, P. 2013. Rolling Horizon Evolution versus Tree Search for Navigation in Single-Player Real-Time Games. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 351–358.

Piette, E.; Soemers, D. J.; Stephenson, M.; Sironi, C. F.; Winands, M. H.; and Browne, C. 2019. Ludii–The Ludemic General Game System. *arXiv preprint arXiv:1905.05013*.

Raimbault, C. 2014. *Colt Express*. Ludonaute.

Robbins, M. 1971. *Uno*. AMIGO.

Serrino, J.; Kleiman-Weiner, M.; Parkes, D. C.; and Tenenbaum, J. 2019. Finding Friend and Foe in Multi-Agent Games. In *Advances in Neural Information Processing Systems*, 1251–1261.

Sfikas, K., and Liapis, A. 2020. Collaborative Agent Gameplay in the Pandemic Board Game. In *International Conference on the Foundations of Digital Games (FDG)*.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the Game of Go without Human Knowledge. *nature* 550(7676):354–359.

Summerville, A. J., and Mateas, M. 2016. Mystical Tutor: A Magic: The Gathering Design Assistant Via Denoising Sequence-To-Sequence Learning. In *Twelfth artificial intelligence and interactive digital entertainment conference*.

Szita, I.; Chaslot, G.; and Spronck, P. 2009. Monte-Carlo Tree Search in Settlers of Catan. In *Advances in Computer Games*, 21–32. Springer.

Teuber, K. 1995. *The Settlers of Catan*. Mayfair Games.

Wrede, K.-J. 2000. *Carcassonne*. Hans im Glück.