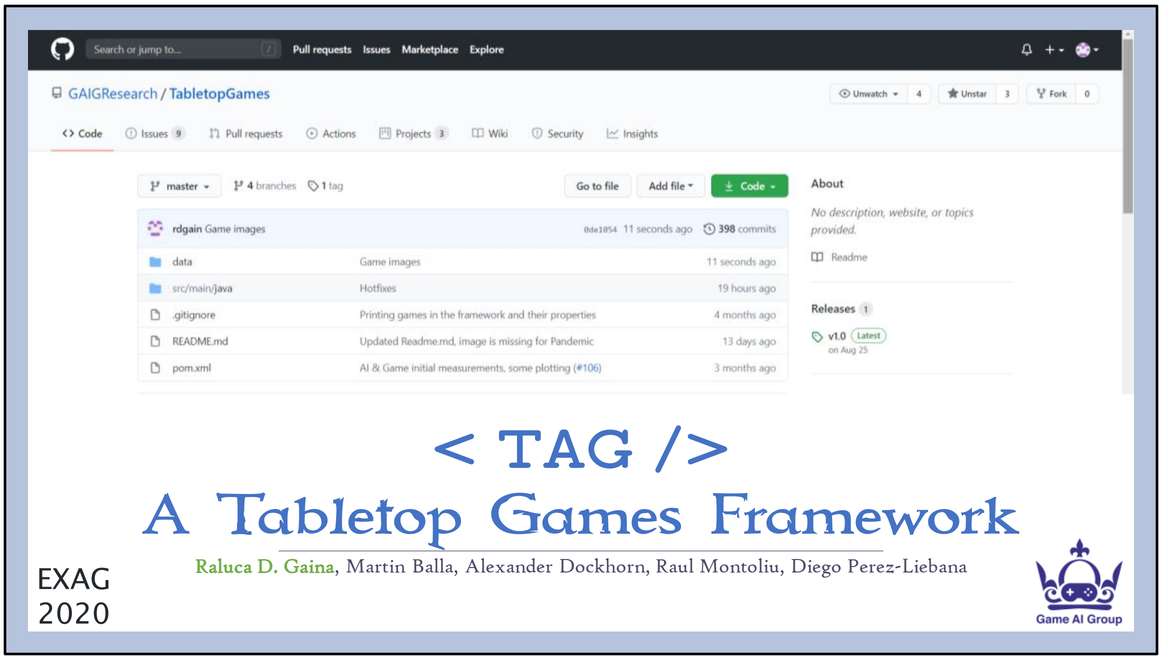


Search or jump to... Pull requests Issues Marketplace Explore

GAIGResearch / TabletopGames

Unwatch 4 · Unstar 3 · Fork 0

Code · Issues 9 · Pull requests · Actions · Projects 3 · Wiki · Security · Insights

master · 4 branches · 1 tag · Go to file · Add file · Code

rdgain Game images · 0de1054 11 seconds ago · 398 commits

data	Game images	11 seconds ago
src/main/java	Hotfixes	19 hours ago
.gitignore	Printing games in the framework and their properties	4 months ago
README.md	Updated Readme.md, image is missing for Pandemic	13 days ago
pom.xml	AI & Game initial measurements, some plotting (#106)	3 months ago

About

No description, website, or topics provided.

Readme

Releases 1

v1.0 Latest on Aug 25

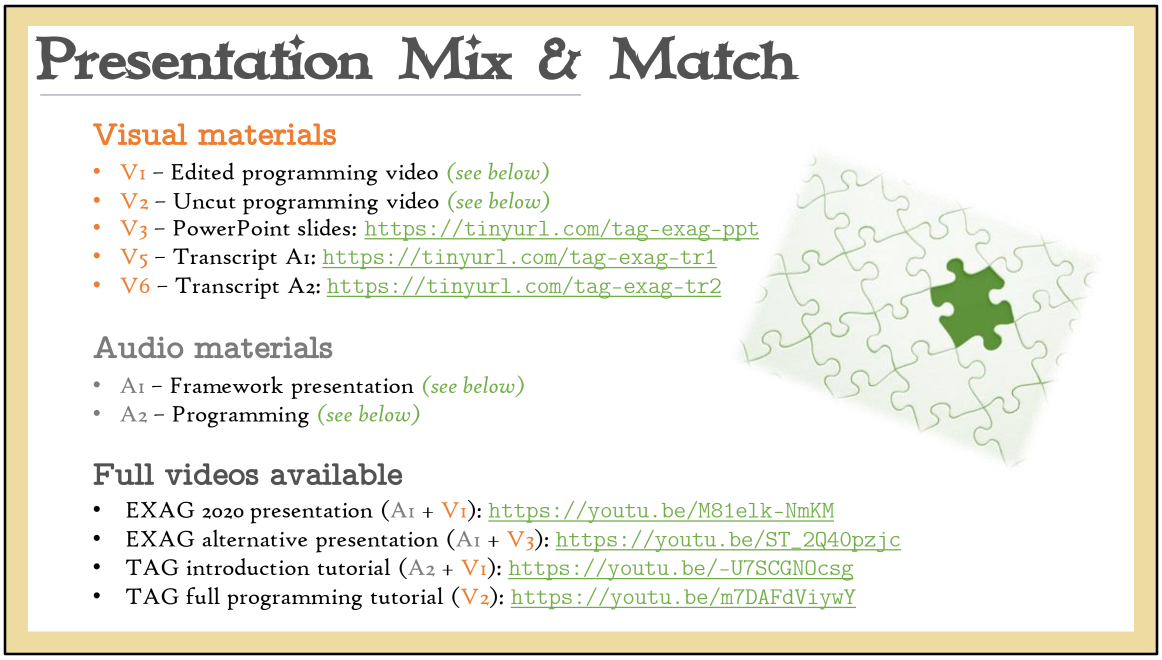
< TAG />
A Tabletop Games Framework

Raluca D. Gaina, Martin Balla, Alexander Dockhorn, Raul Montoliu, Diego Perez-Liebana

EXAG 2020

Game AI Group

Hi there! I am Raluca Gaina, a PhD student from Queen Mary University of London. And this is TAG, an AI framework built around tabletop games. This is an ongoing project started during an internal game AI hack, which continues to be developed and attract contributions from many more researchers and students as I speak!



Presentation Mix & Match

Visual materials

- V1 – Edited programming video *(see below)*
- V2 – Uncut programming video *(see below)*
- V3 – PowerPoint slides: https://tinyurl.com/tag-exag-ppt
- V5 – Transcript A1: https://tinyurl.com/tag-exag-tr1
- V6 – Transcript A2: https://tinyurl.com/tag-exag-tr2

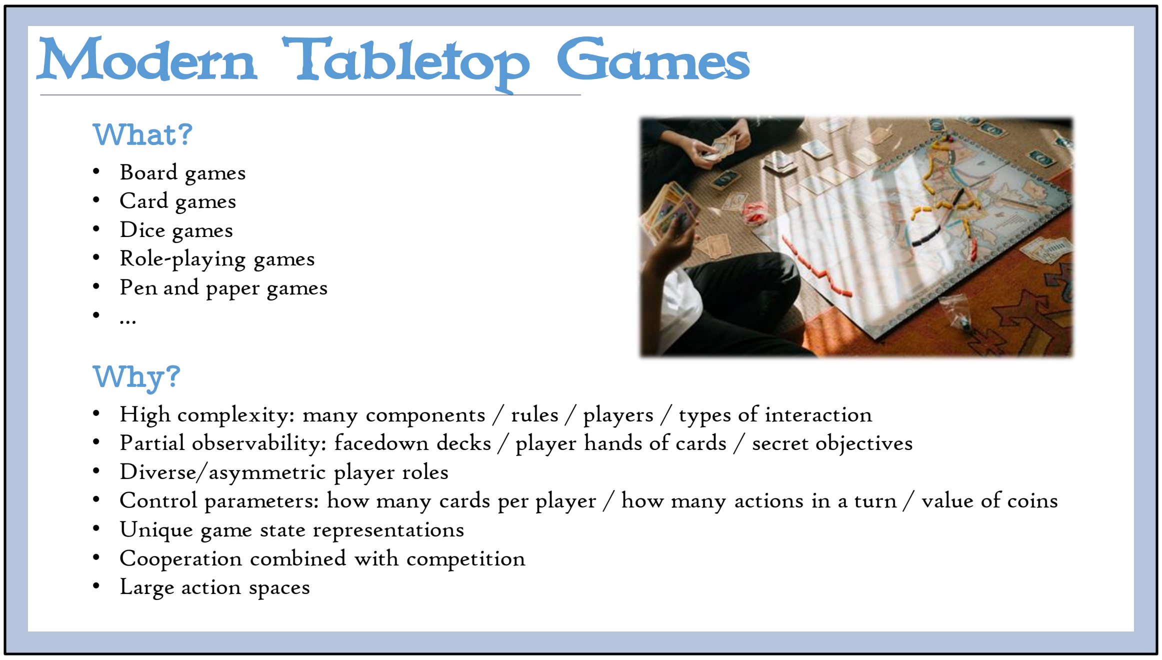
Audio materials

- A1 – Framework presentation *(see below)*
- A2 – Programming *(see below)*

Full videos available

- EXAG 2020 presentation (A1 + V1): https://youtu.be/M81elk-NmKM
- EXAG alternative presentation (A1 + V3): https://youtu.be/ST_2Q40pzjc
- TAG introduction tutorial (A2 + V1): https://youtu.be/-U7SCGNOcsg
- TAG full programming tutorial (V2): https://youtu.be/m7DAFdViywY

I'll talk a bit about this today, but first… given the virtual nature of the EXAG workshop this year, I've prepared several versions of visual and audio material to hopefully keep you all more entertained than just watching me read through some slides. One question raised during the review process was how easy it actually is to implement a game in the framework, so you are watching now the implementation of Dots and Boxes which took about 1 hour in real time, while listening to the framework presentation. All materials and audio transcripts are publicly available for accessibility purposes. In case you prefer my voiceover to match the visuals you're seeing, please check the powerpoint slides linked.



Modern Tabletop Games

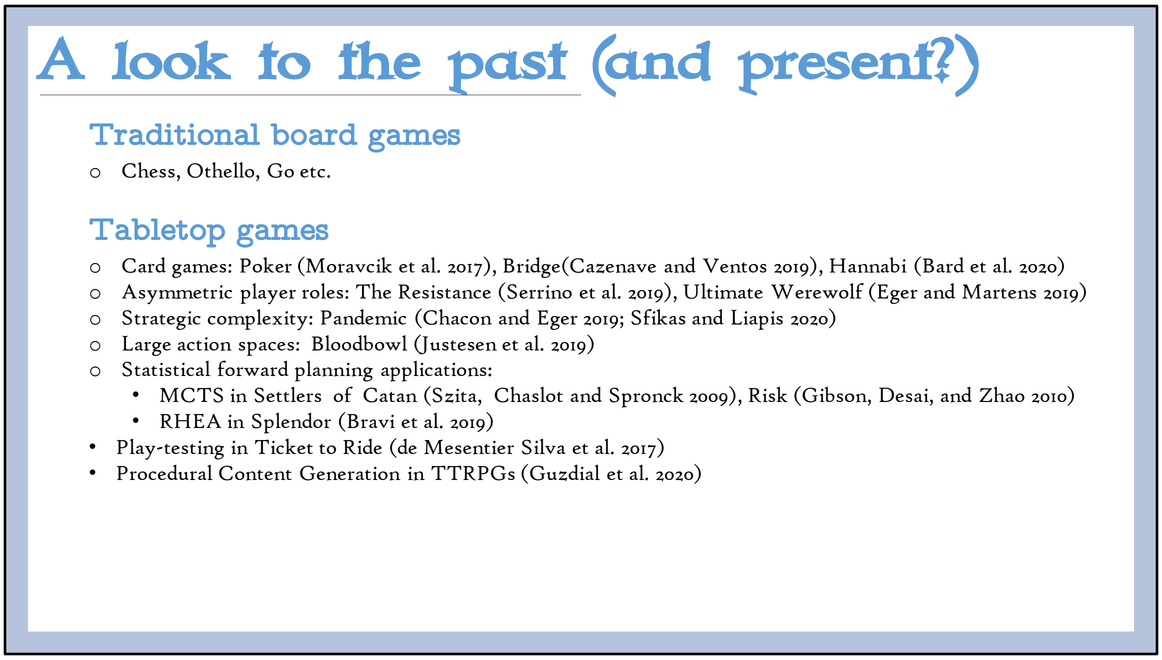
What?

- Board games
- Card games
- Dice games
- Role-playing games
- Pen and paper games
- …

Why?

- High complexity: many components / rules / players / types of interaction
- Partial observability: facedown decks / player hands of cards / secret objectives
- Diverse/asymmetric player roles
- Control parameters: how many cards per player / how many actions in a turn / value of coins
- Unique game state representations
- Cooperation combined with competition
- Large action spaces

The TAG project is all about tabletop games. With this term we mean any sort of analogue game that can be played on a physical surface. Examples include, but are not limited to, board games, card games, dice games, role-playing games and other pen and paper games. And more specifically, we're looking at *modern* tabletop games. This term is harder to define, as the type of games we are interested in were even played in ancient times (so not exactly "modern"?). But, in this context, we refer to those games that are of high complexity: they contain many different types of components (combining cards and dice, for example), many rules and types of interactions between many players. A lot of these games also include partial observability, by hiding parts of the board, or some players' hands, or by placing randomly shuffled decks of cards face down on the table. Players often take on different roles within the game, and each game has a series of parameters that control its flow and balance, such as the number of cards in a player's hand, or the number of events that take place at a specific point in time. These games often feature unique game state representations due to their complex collections of game pieces, may include both cooperative and competitive aspects, between players or between the players and the game, and they often have very large action spaces. All of these make them very interesting problems for diverse Artificial Intelligence methods.


A look to the past (and present?)

Traditional board games

- Chess, Othello, Go etc.

Tabletop games

- Card games: Poker (Moravcik et al. 2017), Bridge(Cazenave and Ventos 2019), Hannabi (Bard et al. 2020)
- Asymmetric player roles: The Resistance (Serrino et al. 2019), Ultimate Werewolf (Eger and Martens 2019)
- Strategic complexity: Pandemic (Chacon and Eger 2019; Sfikas and Liapis 2020)
- Large action spaces: Bloodbowl (Justesen et al. 2019)
- Statistical forward planning applications:
  - MCTS in Settlers of Catan (Szita, Chaslot and Spronck 2009), Risk (Gibson, Desai, and Zhao 2010)
  - RHEA in Splendor (Bravi et al. 2019)
- Play-testing in Ticket to Ride (de Mesentier Silva et al. 2017)
- Procedural Content Generation in TTRPGs (Guzdial et al. 2020)

We've seen the growth of AI techniques closely linked to traditional board games since its very beginnings, with artificial players becoming proficient at Chess, Othello and more recently Go. But research has also looked into other types of tabletop games too. There have been several AI agents playing card games, or games focused around deceit and asymmetric player roles, such as The Resistance and Ultimate Werewolf. AI players have started attacking games of higher strategic complexity, such as Pandemic, or with large action spaces, such as Bloodbowl. And we've seen specific applications of statistical forward planning methods in Settlers of Catan, Risk and Splendor. What's special or interesting to us about these cases is that statistical forward planning methods require a model of the game in order to perform simulations of what might happen in different situations – models which can become large and slow the more complex a game is, yet allow these powerful AI methods to play games to a high level of performance. But these models are also an additional programming step which can become tedious to implement.

But AI is not only used for playing these games anyway. We can see applications of AI for play-testing in Ticket to Ride, meant to find bugs, balancing issues or exploits, which usually become apparent only after extensive human play-testing. And we've seen the relationship between tabletop role-playing games and Procedural Content

Generation drawn by Matthew Guzdial and others quite recently.

A look to the past (and present?)

Description-language-based frameworks

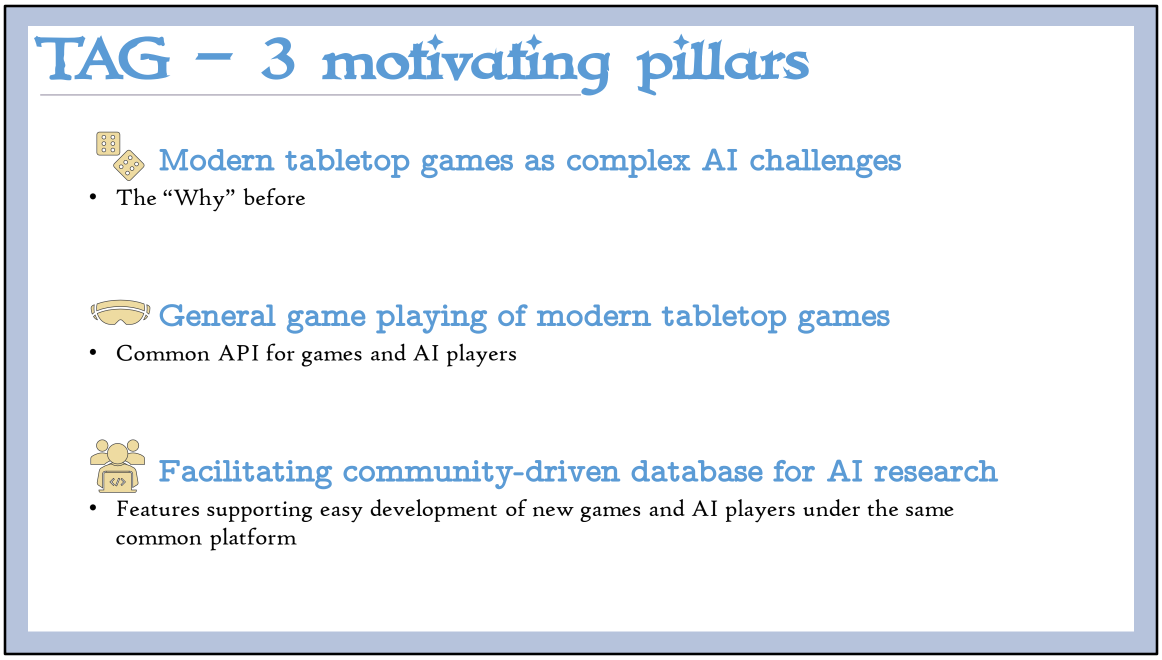
- General Game Playing (GGP) (Genesereth, Love, and Pell 2005)
- Ludii (Piette et al. 2019)
- Regular boardgames (RBG) (Kowalski et al. 2019)

Freeform frameworks

- Tabletop Simulator (Henry 2015)
- OpenSpiel (Lanctot et al. 2019)

Some of this work was brought together into commonly used frameworks, featuring a collection of different games and AI players for easy benchmarking of new methods. Some of these include the General Game Playing framework and competition from all the way back in 2005, and more recently the Ludii and Regular Boardgames frameworks. All of these describe their games in a framework-specific language: GGP uses the game description language, which describes the state of a game as a series of facts, and the game mechanics as logical rules. Ludii uses a series of "ludemes" (or game logic units) which together make up rules or state descriptions. And RBG uses regular expressions for improved efficiency.

At the other end of the spectrum, there are some freeform frameworks which allow developers to build games with intuitive tools or complete programmatic control, such as the Tabletop Simulator and OpenSpiel. In these cases, adding games becomes much simpler, as users do not have to learn a framework specific language and all of its intricacies; and even more, the limits which might exist in the framework's description language can be overcome here, as almost anything becomes possible, within the limits of the user's technical abilities.



TAG – 3 motivating pillars

Modern tabletop games as complex AI challenges

- The "Why" before

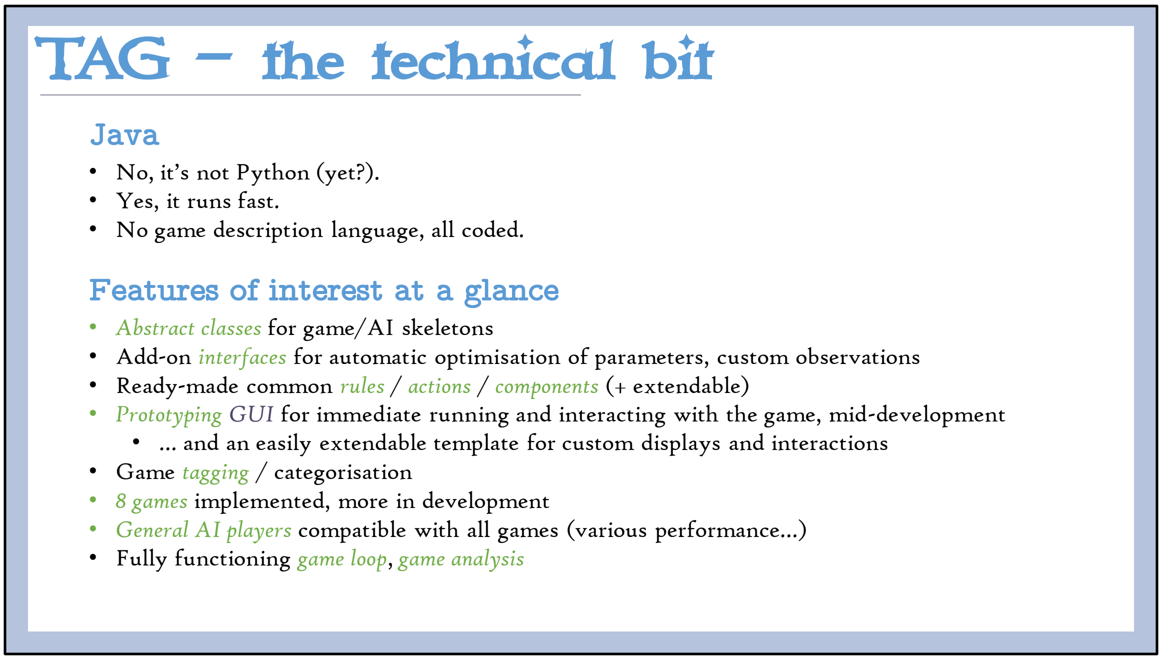
General game playing of modern tabletop games

- Common API for games and AI players

Facilitating community-driven database for AI research

- Features supporting easy development of new games and AI players under the same common platform

We take the best of these approaches forward to build our own framework, with 3 main contributions targeted. First, we bring into focus the challenges brought by modern tabletop games for AI methods. Second, we promote research into general artificial intelligence, by providing a common API for games and AI players, allowing artificial players to try to solve an ever increasing collection of problems. And third, we provide features to facilitate the development of new games and AI players under the same common platform, and invite community contributions to drive research forward.



TAG – the technical bit

Java

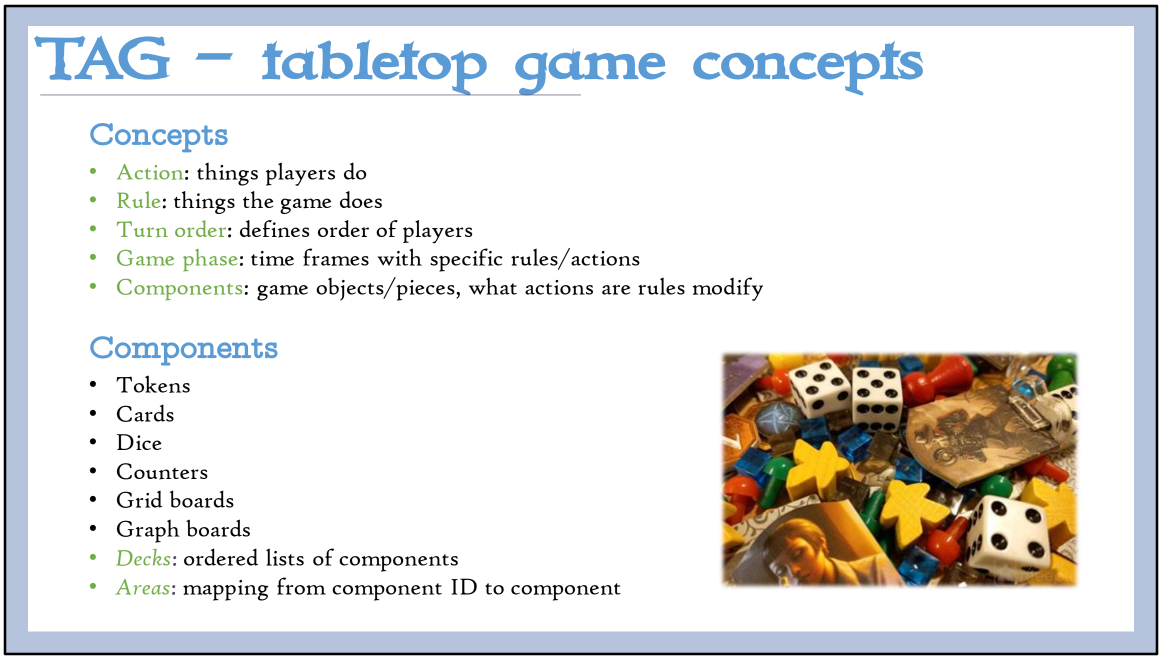
- No, it's not Python (yet?).
- Yes, it runs fast.
- No game description language, all coded.

Features of interest at a glance

- *Abstract classes* for game/AI skeletons
- Add-on *interfaces* for automatic optimisation of parameters, custom observations
- Ready-made common *rules* / *actions* / *components* (+ extendable)
- *Prototyping GUI* for immediate running and interacting with the game, mid-development
  - … and an easily extendable template for custom displays and interactions
- Game *tagging* / categorisation
- *8 games* implemented, more in development
- *General AI players* compatible with all games (various performance…)
- Fully functioning *game loop*, *game analysis*

But let's dive a bit deeper into what the framework actually does. TAG is built in Java, taking advantage of the execution speed of this language and several AI players already available and provided with the code. We do not use a game description language, everything is pure code, aiming to speed up development and execution, as well as lower the barrier of entry for new users. The core of the framework is made up by several abstract classes, providing generic functionality and a simple to follow skeleton for implementing both the games and the AI players. Several classes can further make use of additional interfaces to allow compatibility with other systems included, such as automatic optimisation of parameters or custom observations. We provide several ready-made rules, actions and components which are common across a wide variety of games, all of which are easily extendable.

We include a graphical user interface compatible with any game – allowing to run it as soon as the two main skeleton classes are created, the game state and the forward model. TAG further includes a fully functioning game loop, game tagging / categorisation for easy filtering through the games collection and game analysis. We currently have 8 games implemented and several general AI players compatible with all games – although compatibility does not equal proficiency in all cases.



TAG – tabletop game concepts

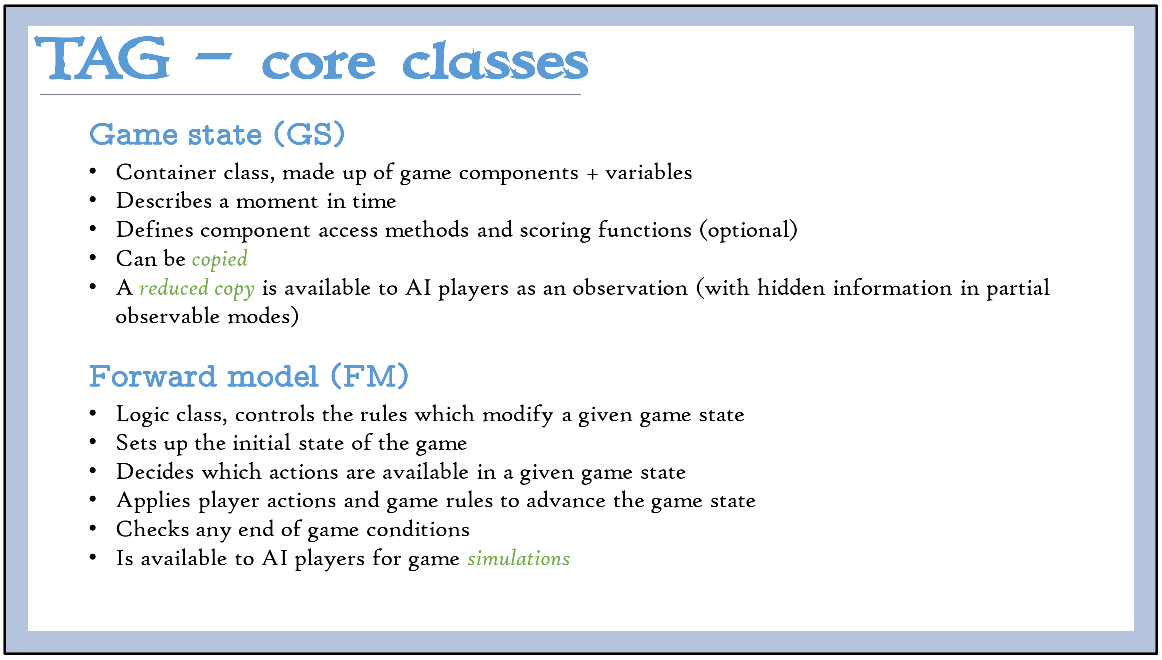
Concepts

- Action: things players do
- Rule: things the game does
- Turn order: defines order of players
- Game phase: time frames with specific rules/actions
- Components: game objects/pieces, what actions are rules modify

Components

- Tokens
- Cards
- Dice
- Counters
- Grid boards
- Graph boards
- *Decks*: ordered lists of components
- *Areas*: mapping from component ID to component

We use several concepts common in tabletop games to define the core of a game. Actions are things players can do in the game. Rules are things the game does, outside of a player's action. Turn orders define the order in which the players are going to be asked for actions. Game phases are time frames in which specific rules apply, or the players have specific actions available. And each game is made up of several components (or game pieces), which is what actions and rules modify to change the state of the game. Components currently supported include tokens, cards, dice, counters, grid and graph boards, all with different properties and functions associated. We can also group these into collections: decks are ordered lists of components, and areas are maps, mapping from the component ID to the component itself. Both areas and decks are considered components themselves and can be stacked (for example, an area may contain several decks).



TAG – core classes

Game state (GS)

- Container class, made up of game components + variables
- Describes a moment in time
- Defines component access methods and scoring functions (optional)
- Can be *copied*
- A *reduced copy* is available to AI players as an observation (with hidden information in partial observable modes)

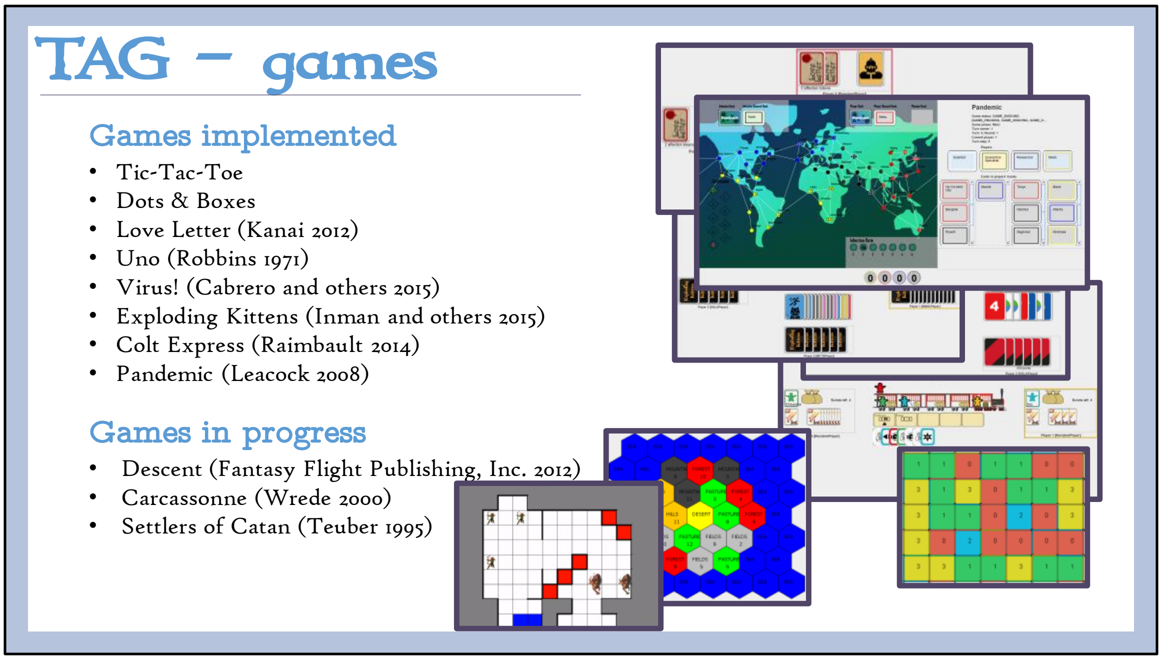
Forward model (FM)

- Logic class, controls the rules which modify a given game state
- Sets up the initial state of the game
- Decides which actions are available in a given game state
- Applies player actions and game rules to advance the game state
- Checks any end of game conditions
- Is available to AI players for game *simulations*

In order for a game to be created, two main classes are required: a game state, and a forward model.

The game state is a container class, made up of game components, and any other variables that can be used to speed up execution. A game state describes a moment in time, and defines component access methods and, optionally, a scoring function to be used by the AI players. A game state can be *copied*, and AI players receive such a copy, reduced to only include the information they can actually observe at the time.

The forward model contains the logic of the game, including all of the rules that modify a given game state. It sets up the initial state of the game, decides which actions are available for the players, applies player actions and game rules to advance the game state and checks any end of game conditions. The forward model is available to AI players for game simulations.



TAG – games

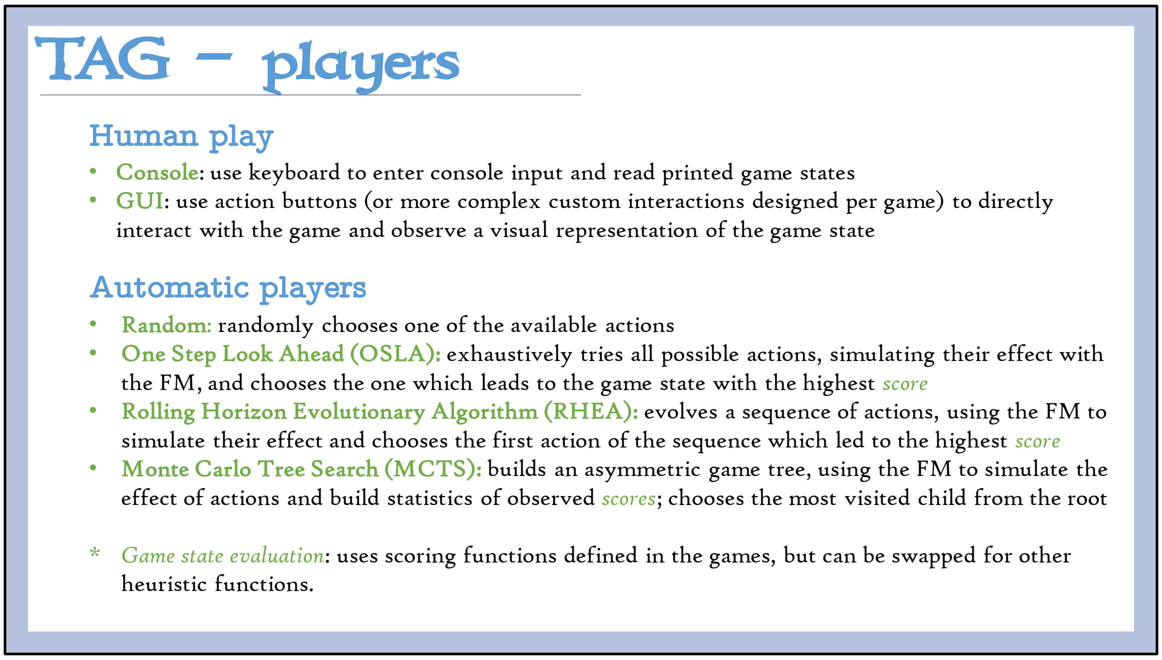
Games implemented

- Tic-Tac-Toe
- Dots & Boxes
- Love Letter (Kanai 2012)
- Uno (Robbins 1971)
- Virus! (Cabrero and others 2015)
- Exploding Kittens (Inman and others 2015)
- Colt Express (Raimbault 2014)
- Pandemic (Leacock 2008)

Games in progress

- Descent (Fantasy Flight Publishing, Inc. 2012)
- Carcassonne (Wrede 2000)
- Settlers of Catan (Teuber 1995)

These are the games currently implanted in the framework – starting from very simple instances, such as Tic Tac Toe and Dots & Boxes, to more complex games like Pandemic and Colt Express. We also have several games currently in progress, including Descent, Carcassonne, and Settlers of Catan, the last of which should be ready very soon.



TAG – players

Human play

- Console: use keyboard to enter console input and read printed game states
- GUI: use action buttons (or more complex custom interactions designed per game) to directly interact with the game and observe a visual representation of the game state

Automatic players

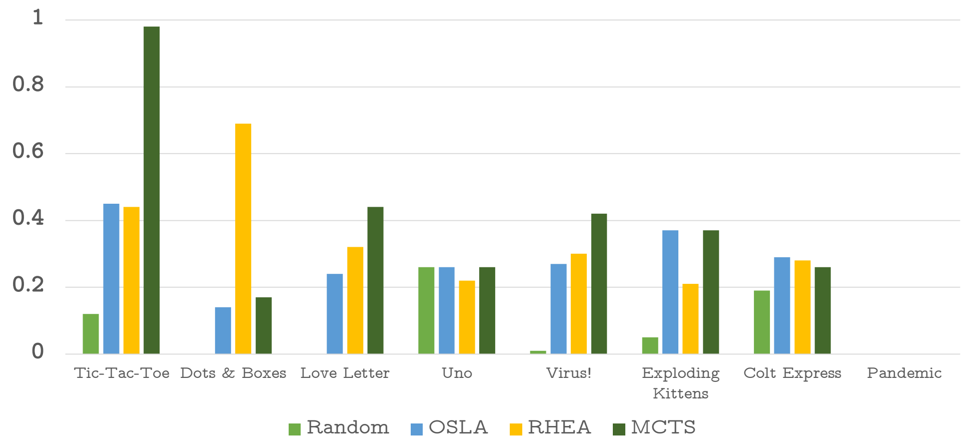
- Random: randomly chooses one of the available actions
- One Step Look Ahead (OSLA): exhaustively tries all possible actions, simulating their effect with the FM, and chooses the one which leads to the game state with the highest *score*
- Rolling Horizon Evolutionary Algorithm (RHEA): evolves a sequence of actions, using the FM to simulate their effect and chooses the first action of the sequence which led to the highest *score*
- Monte Carlo Tree Search (MCTS): builds an asymmetric game tree, using the FM to simulate the effect of actions and build statistics of observed *scores*; chooses the most visited child from the root

* *Game state evaluation*: uses scoring functions defined in the games, but can be swapped for other heuristic functions.

To actually play the games, you can do so as human players in two ways: via the console (which would print out the game state and ask for the index of the next action out of the suggested possibilities), or via the graphical user interface (which by default includes buttons for all the possible actions and dragging of components on the screen, but may include customized interactions, such as selecting specific game pieces for filtered action lists).

Additionally, we have four artificial players that can be run with any game: random takes a random action out of all possible. One step look ahead greedily selects the action which leads to the next best state. The rolling horizon evolutionary algorithm evolves a sequence of actions and chooses the first action of the best sequence to execute at the end, and monte carlo tree search builds and searches the game tree for the next best move. OSLA, RHEA and MCTS all use the forward model to simulate possible future scenarios, and an evaluation function to evaluate the game state, which can be set to custom functions.

AI Player Performance

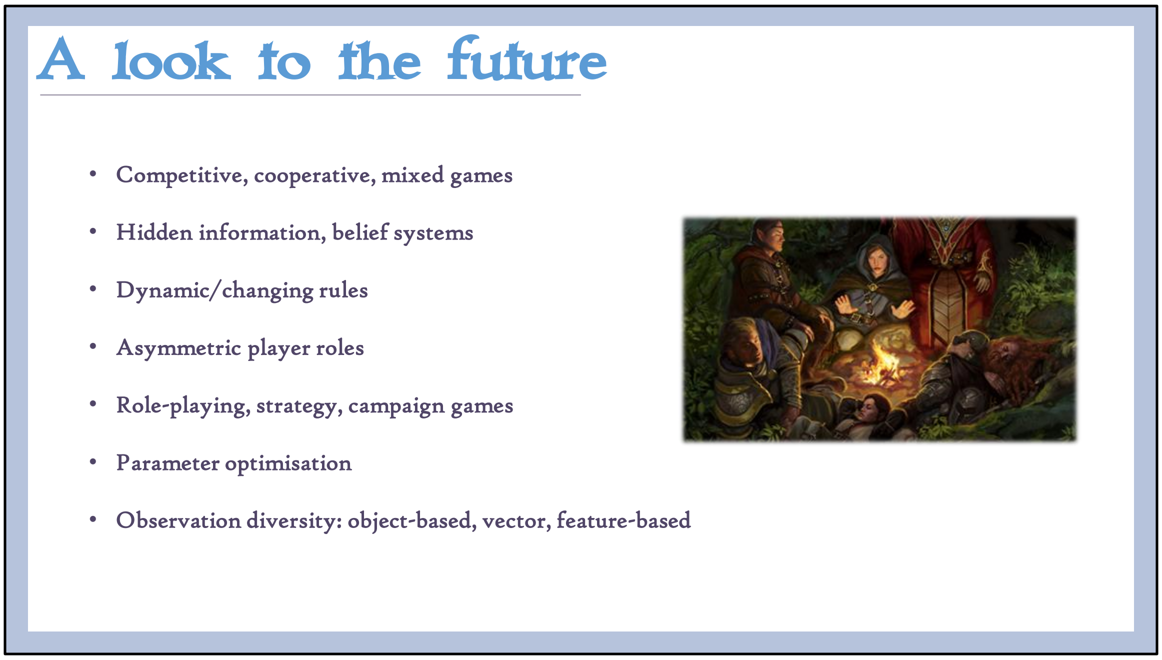


We’ve run our agents on the current collection of games. For Tic Tac Toe, which is a 2-player only game, we’ve run a round robin tournament, where each agent played 100 times against all other agents. In Pandemic we’ve run teams of the same agent type 100 times each. For all other games, we’ve run 100 4-player games with 1 instance of each player. For the search-based algorithms, we used a budget of 4000 forward model simulations at each decision making game tick.

Results generally indicate MCTS to be the best agent, with the highest win rate in 5 out of 7 competitive games (thus excluding Pandemic) and an overall win rate of 41% in these games. OSLA also appears to beat RHEA in some of the games, which is potentially due to the large uncertainty built up in its rigid sequences of actions (as opposed to the more flexible game trees of MCTS), with the greedy approach appearing to be preferable in some games, especially those simple enough or with a heuristic function accurate enough (such as Tic Tac Toe). However, we do observe RHEA to clearly dominate all other agents in Dots & Boxes, a game not included in earlier paper publication, but added for this presentation. This is an interesting case to be analysed in more details.

We also note that no agent is able to win in Pandemic. Stronger heuristics and better analysis, such as the distance from the winning game states for each AI team, could lead to improved performance or show interesting insights into the agents' behaviours and abilities.

Additionally, some games (Uno and Colt Express) see very close performance for all players, including random. This highlights the difficulty of the problems proposed, as well as the importance of the heuristic chosen for a game, as some features of a game state may prove deceiving (such as having more cards in hand in Uno). However, it is worth noting that the search methods presented were tested in their vanilla form, but they benefit from ample literature and large parameter spaces, which can be tuned for increased performance.

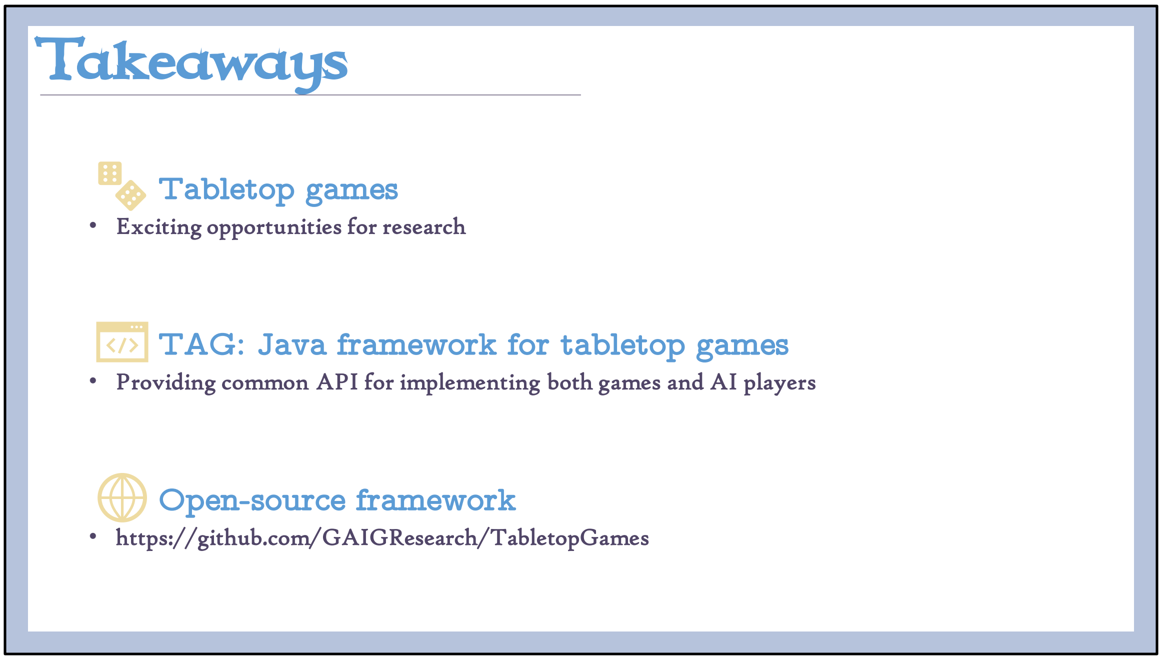


A look to the future

- Competitive, cooperative, mixed games
- Hidden information, belief systems
- Dynamic/changing rules
- Asymmetric player roles
- Role-playing, strategy, campaign games
- Parameter optimisation
- Observation diversity: object-based, vector, feature-based

Lastly, I want to take a look at the future, and the opportunities for research opened up through this project. There are lots more details in the paper and I don't have time to go through all of them in detail. But I do want to highlight two things here: the first, is role-playing games – this is a very difficult challenge, as it goes beyond tactics and playing the correct actions, towards even finding which actions are possible or make sense in a situation, and building a consistent, engaging story together with the other players. Plus, these games often feature a dungeon master or game master, whose job is not to win necessarily, but to give the other players an interesting experience. How would an AI even go about that? The ongoing implementation of Descent in our framework aims to be a step in this direction.

And second, I'd like do briefly highlight that the framework offers functionality for easily optimising parameters of both AI players and games – looking for a better balanced version of your favourite game? TAG could put that together.



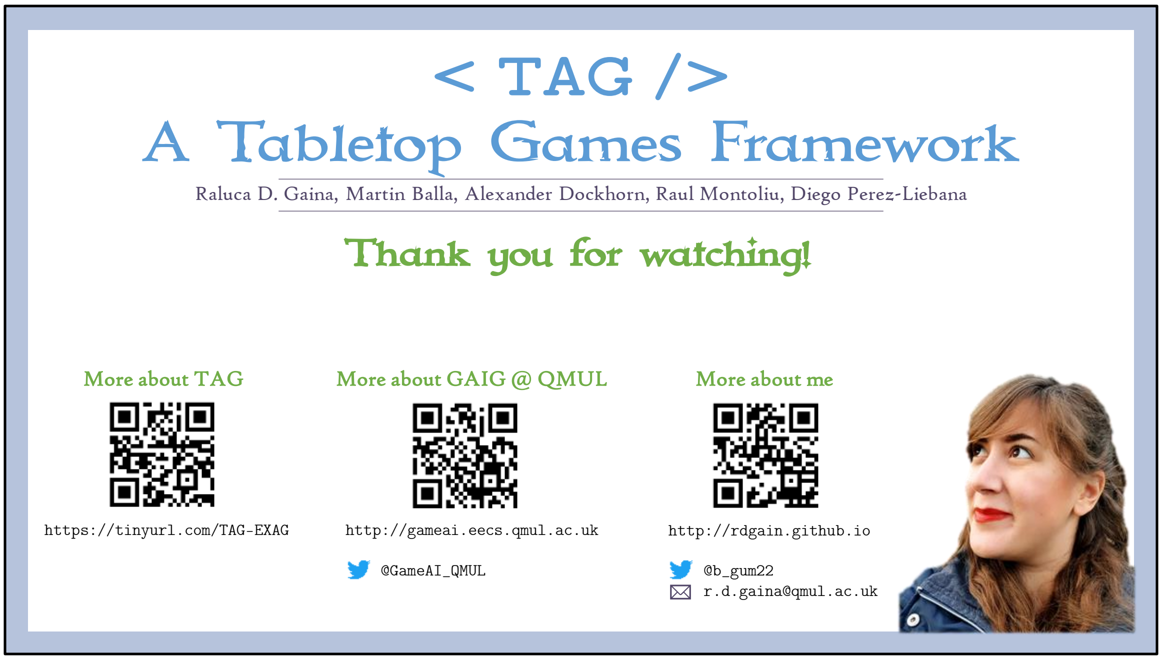
Takeaways

Tabletop games
- Exciting opportunities for research

TAG: Java framework for tabletop games
- Providing common API for implementing both games and AI players

Open-source framework
- https://github.com/GAIGResearch/TabletopGames

And if you've zoned out through some of this presentation, please take these 3 things away! First, that tabletop games present so many exciting opportunities for research – go play one and think how an AI method would do something happening in that game! Second, that I've been talking about our TAG project, which is all about building a collection of tabletop games and AI players able to play them all. Third, check it all out yourself – download the GitHub repository, have a play, and give us feedback, or even start implementing your favourite game today!



< TAG />
A Tabletop Games Framework

Raluca D. Gaina, Martin Balla, Alexander Dockhorn, Raul Montoliu, Diego Perez-Liebana

Thank you for watching!

More about TAG	More about GAIG @ QMUL	More about me
https://tinyurl.com/TAG-EXAG	http://gameai.eecs.qmul.ac.uk	http://rdgain.github.io
	@GameAI_QMUL	@b_gum22
		r.d.gaina@qmul.ac.uk

Thank you very much for watching! If you'd like, you can read more about TAG, find more details about our Game AI group at Queen Mary University of London and all of the exciting things we get up to, or get in touch with me through the links on the screen. Have a great rest of the day!