TDNTupleAgt.docx TODO.docx notes-WK-RubiksCube.docx

FUTURE WORK FOR GAME LEARNING – WK/08/2023

The following describes ideas for student projects and student theses in the context of GBG, see https://github.com/WolfgangKonen/GBG and https://blogs.gm.fh-koeln.de/ciop/research/gbg-general-board-games/. It is an internal document.

OPEN AND PARTLY OPEN PROJECTS

Note that the sections *New Agents* and *New Games* contain **several** open and partly open projects in their lists. Some of them are further worked out in the sections below.

See further down for the section of closed projects. It might be useful to read them, as they contain building blocks for the open projects and as they might give room for extension in one way or the other.

NEW AGENTS AND LEARNING STRATEGIES

- **CNN-like agent**
- **Java-Python bridge for DL agents** (DL: deep learning)
- Language-independent agent: An agent like in microRTS, "Creating non-Java AIs", which can be in any non-Java language and corresponds with a server via socket connections
- Agent Balancing or **Game Balancing through Tournament Selection**.
- **Replay Experience Buffer** (a buffer which collects data from the latest episodes to have good data and enough data for training). *(partly done by Julian Cöln)*
- **Kociemba's Solver for Rubik's Cube**
- **Transfer learning** *(theme needs to be worked out)*

NEW GAMES

Games of special interest for GBG:

- **Investigations on the learnability of Sim in GBG**
- **Investigations on the learnability of Nim3P in GBG**
- **Rubik's Cube Visualization** (**Rubik's Cube** *partly done by own work [Konen2023a]*)
- Shut The Box, https://en.wikipedia.org/wiki/Shut_the_Box, a 2-, 3-, …, N-player nondeterministic dice game. Drawback: There is perhaps not much strategy to learn.
- **Poker**, https://de.wikipedia.org/wiki/Poker, in der Texas Hold'em-Variante – because it can be played with 2 to up to 10 or 14 players and is quite different from other board games. See also recent DeepStack-arXiv-publication. Drawback: quite complex rules and needs long tournaments to determine winner. *(partly done by Tim Zeh)*
- **Blackjack** – similar to Poker, but a bit simpler *(partly done by Simon Meissner)*
- **Stochastic RL algorithms Python for BlackJack & (Kuhn) Poker** *(see RL3-document)*
- **Yavalath extensions**.
- Are there other interesting N-player games with N>2? The problem is that some games with N>2 do not have any longer a true winning strategy, it is not possible to say: "This is the best move in that state".

Further 2-person games (if students are interested):

- Mühle, https://de.wikipedia.org/wiki/Mühle_(Spiel)
- Checkers (Dame), https://de.wikipedia.org/wiki/Dame_(Spiel)
- Qubic (3D-TicTacToe)
- Abalone
- LoA (Lines of Action)
- Yavalath (the first computer-generated game) *(partly done by Ann Weitz)*
- Other imperfect information games (Stratego, DarkHex?)
- … and perhaps other interesting games from Ludii.

BASELINE ASPECTS FOR GAMING

- **Client-Server Framework for GBG**
- **Game Balancing**

CNN-LIKE AGENT

GBG has seen a number of successes: The TD n-tuple agent works very well on smaller boards in various games (2048, Connect Four, Nim, smaller Hex boards) But TD n-tuple reaches a barrier when it comes to larger boards (Hex boards 7x7 and higher, Othello). This is probably because the number of weights needed for larger boards grows exponentially.

A weight-saving variant would be to use the main idea from **CNN (Convolutional Neural Networks)**: Define weight-sharing filters or weight-sharing n-tuples, which have trainable weights but share them with filters placed at another position on the board.

Work steps:

- Elaborate on the concept which is only sketched here.
- Research similar literature
- Option 1: Implement CNNs in Java: The library DeepLearning4J may be something to look at.
- Option 2: Work with a Java-Python bridge and use the possibility to build flexible CNN architectures from TensorFlow, Keras or similar.
- Test the agents on Hex 7x7 and higher or on Othello

Prior knowledge:

- Java [, Python, TensorFLow, Keras]
- CNN and neural networks in general

JAVA-PYTHON BRIDGE FOR DL AGENTS

JAVA-PYTHON BRIDGE FOR DEEP REINFORCEMENT LEARNING

In the current GBG framework, the neural nets are shallow or very simple networks. If we want to use deep reinforcement learning (DRL) techniques, it might be desirable to do the time-consuming training with the help of StableBaselines3 (SB3, a very good collection of stable DRL algorithms, https://stable-baselines3.readthedocs.io/en/master) in Python.

This requires a bridge between Java and Python. There are several such bridges, two of them are **Py4J** (www.py4j.org) and **PyJNIUS** (https://pypi.org/project/pyjnius). A prior project by Niklas Fabig [Fabig21] for another application shows a good working example.

Work steps:

- Become familiar with Py4J, PyJNIUS and GBG.
- Design one (or perhaps several alternative) concept(s) for Java-Python bridge: Which tasks to hand over, with which granularity?
- First a concept for one specific game, but later with the possibility to work for arbitrary games
- Implement these concepts in GBG
- The minimum goal is to select one algorithm from SB3 and train with it an agent on a simple game from GBG, e.g. TicTacToe.
- The desirable goal is to test the bridge on various games, various SB3 algorithms, measure speed, evaluate quality.

Prior knowledge:

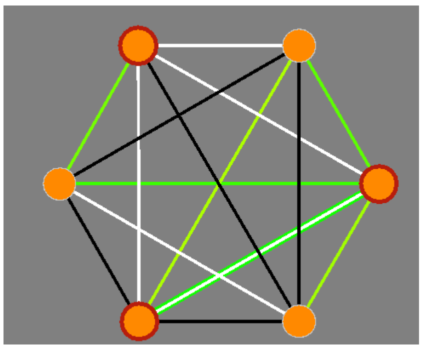
- Java
- Python
- Knowledge of RL and SB3 helps, but can be also acquired on-the-job

JAVA-PYTHON BRIDGE FOR CNN

Once the Java-Python bridge from the previous project is established, we can use it also to replace the DRL agent with a CNN agent, using flexible Tensorflow or Keras algorithms on the Python side.

See project "CNN-like Agent", Option 2.

LEARNABILITY OF SIM IN GBG



Sim (https://en.wikipedia.org/wiki/Sim_(pencil_game) or https://de.wikipedia.org/wiki/Sim_(Spiel)) is an interesting game for several reasons:

(a) It has simple rules,
(b) it has quite a complex winning strategy,
(c) it is a scalable game (# of nodes),
(d) it has a large number of symmetries (in fact, exponentially growing with # of nodes),
(e) it can be played with 2 or 3 players (3 separate players or 1-vs-2 variant).

Sim is now available in GBG, but topics around the **learnability of Sim** do need further research:

- Which agents with which parameters can learn to play best: in 6-, 7-, …, 17-nodes Sim and with 2 or 3 players (17 nodes would be the first game without a tie in the 3-player variant).
- Symmetries normally make a game simpler to learn. But Sim has so many symmetries (720 for 6-node Sim, and exponentially rising with higher node numbers) that the normal strategy for symmetries: "Take all of them" does not work any longer. What are better strategies?
- Is there a lower node number than 17 where the 1-vs-2 variant has a clear winner?

LEARNABILITY OF NIM/NIM3P IN GBG

Nim3P is now available in GBG. Nim3P is a **3-player variant** of the well-known game Nim.
Nim (https://de.wikipedia.org/wiki/Nim-Spiel, https://en.wikipedia.org/wiki/Nim) is an interesting game for several reasons:

(a) it is a scalable game (# of heaps, size of heaps),
(b) it is simple to learn, if # and size of heaps is small, but more difficult to learn, if they are not,
(c) it has a large number of symmetries (in fact, exponentially growing with # of heaps),
(d) it can be played with 2 players (normal Nim) or 3 players (Nim3P).

Nim3P, the 3-player variant, is briefly covered in the recent publication [Konen2020b] of our research group.

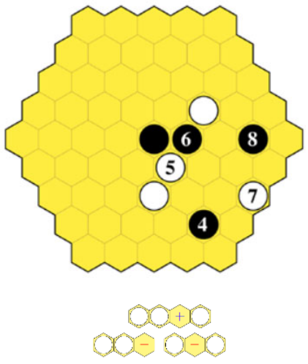
It is not yet fully researched which variants and reward schemes of Nim3P are meaningful such that such a 3-player Nim has a clear winning strategy.
Topics around the **learnability of Nim3P** need further research:

- What variants and reward schemes of Nim3P allow for a clear winning strategy?
- Which agents from GBG can learn to play which scalable Nim3P games well? With which agent-specific parameters?
- Symmetries normally make a game simpler to learn. But Nim and Nim3P has so many symmetries (n! for a game with n heaps) that the normal strategy for symmetries: "Take all of them" does not necessarily work any longer. What are better strategies?

YAVALATH EXTENSIONS

Yavalath (https://de.wikipedia.org/wiki/Yavalath) is the first game completely generated by a computer (evolved by a genetic algorithm, which was designed by Cameron Browne).

In 2021-22, this game has been implemented in GBG by Ann Weitz in her Bachelor thesis. [Weitz2022b]. Here she did first steps towards a Yavalath AI in GBG: The GBG agent with random n-tuples and TD learning applied to the standard Yavalath board (edge length 5) reached a decent winning strength (>80%) against a Ludii MCTS agent, which is already a strong player.



Several extensions remain open for further research:

- Further parameter tuning (number and size of n-tuples, TD parameters)
- Different board sizes
- 3-player variant
- Fixed n-tuples: Would it be beneficial if we use carefully designed and fixed n-tuples instead of random n-tuples? Can we learn something from the shape of n-tuples that are good or bad?
- Coupling of TD-n-tuple-agent with the available MCTS wrapper (AlphaGo-Zero-inspired): Does it boost the playing strength?
- Transfer Learning in Yavalath (see also next topic)

TRANSFER LEARNING

This theme needs still to be worked out

Can we transfer knowledge that an agent has from one game or game variant to another game or game variant? E.g., if we know how to play Hex4, does this help to learn Hex5 or Hex6?

Compare with the work already done in [Soemers21b])

RUBIK'S CUBE VISUALIZATION

Rubik's cube (https://de.wikipedia.org/wiki/Zauberw%C3%BCrfel) is so famous that it probably does not need to be introduced. The state space complexity $4.3*10^{19}$ is enormous.

See Rubik's Cube for the current status of the Rubik's cube subproject in GBG.

The purpose of this project: Integrate a nice **cube visualization** into the project. A good starting point may be **CubeTwister** from http://www.randelshofer.ch/, which offers Java and JavaScript code for various cubes.

Work steps:

- Understand the existing cube implementation in GBG.
- Familiarize with various existing 3D cube visualization (e.g. from http://www.randelshofer.ch/).
- Are extensions necessary / desirable?
- Make a concept on how to integrate an existing cube visualization in GBG or how to set up an own 3D cube visualization
- Implement and test the cube visualization in GBG.
- Evaluate various aspect of the cube visualization (computational resources, ease of use, …)

Possible starting points:

- http://www.randelshofer.ch/ Virtual Cube and CubeTwister→ then CubeTwister and from there to https://github.com/wrandelshofer/VirtualCube (**JavaScript source code**) or https://github.com/wrandelshofer/CubeTwister (**Java source code**).
- https://github.com/davidwhogg/MagicCube: Python-based cube interaction & cube visualizer

KOCIEMBA'S SOLVER FOR RUBIK'S CUBE

The goal of this project is to make Kociemba's solver for Rubik's Cube available as a near-perfect benchmark agent for RubiksCube in GBG.

Kociemba's solver, available as Python package, always finds a solution for a scrambled cube (3x3x3 QTM), often in a few seconds, mostly with a number of moves not exceeding God's number.

Work steps:

- Familiarize with the Python package in https://github.com/hkociemba/RubiksCube-TwophaseSolver
- Make a concept how to interface this with Java-based GBG
    - using a Java-Python bridge (see here for options)
    - using a client-server approach (see Kociemba's README)
- Translate cube configurations from GBG to Kociemba and solutions from Kociemba to GBG
- Implement the interface
- Test and evaluate the resulting software
- Optional: Compare with solutions from GBG's agents [Konen2023a].

CLIENT-SERVER FRAMEWORK FOR GBG

Currently, GBG is a stand-alone program. Several aspects would be better in a client-server architecture

- A server offers a set of game-competitions and user may connect to such competitions and let their AI agents play from a remote client
- A server offers a set of games to be played, a set of pre-trained agents. The user may play from a client against a selected agent in a selected game. Server may record the games played → tournament, analysis, hall of fame, ….

This leads to the following goals for a thesis project *Client-Server Framework for GBG*:

- Make a concept for such a client-server (CS) framework. What parts on client, what on server?

- Make a migration path from the current GBG software to the client-server-GBG. Are serious changes in GBG software needed? Can we keep both versions, standalone and CS in one program or do we have to fork?
- Implement and test CS framework
- A starting point can be the Sim(Hexi) CS implementation (which will be, however, not runnable on most machines, since Java Applets are forbidden): https://www.dbai.tuwien.ac.at/proj/ramsey/source.htm

GAME BALANCING THROUGH TOURNAMENT SELECTION

For some games, the available agents are way too strong against humans. To have a better play experience as a human, a game balancing would be desirable.

Based on the existing Competition Framework for GBG with Elo- and Glicko-rating:

- Generate for game X (many) different agents: select agent types, choose different parametrizations, different wrap levels
- Sort the agents by a tournament on game X to establish the Elo/Glicko rating of all agents.
- This should allow a suitable game balancing (to be worked out)

Tournament selection is just one option to do game balancing, other options can be proposed and tested.

CLOSED PROJECTS

- **AlphaGo-Zero-inspired GBG agent** *(done by Johannes Scheiermann)*
- **Ludii Interface for GBG** *(done by Ann Weitz, Meltem Seven, Johannes Scheiermann)*
- **Othello vs. Edax** *(done by Johannes Scheiermann)*
- **Competition Framework for GBG** *(done by Felix Barsnick)*
- **GBG-based agents for Ludii**
    - Reinforcement-Learning GBG agents for Ludii *(partly done by Johannes Scheiermann and Ann Weitz, but missing: the strength comparison, Nim, Hex & Connect-Four)*
    - Perfect- or strong-playing agents for Ludii *(now done within the Bachelor thesis of Meltem Seven, including the Ludii interface for Nim & Connect-Four and the strength comparison for various agents)*

GBG-BASED AGENTS FOR LUDII

The following two projects have already been done (PP and Bachelor thesis Meltem Seven 2023), but **perhaps extensions are possible in one way or the other**.

REINFORCEMENT-LEARNING GBG AGENTS FOR LUDII

Ludii (https://ludii.games/) is an interesting game platform, developed by Cameron Browne and his team, with a large variety of games and with a number of different tree-based agents (MCTS, Alpha-Beta, no pre-training). It offers the possibility to play with a user-supplied JAR-agent against the Ludii agents. Ludii does not offer the possibility to train **reinforcement learning (RL) agents**. It would be interesting to see, how RL agents for Connect Four or Othello, that were trained in GBG and packed into a JAR, compare with the built-in Ludii agents on the Ludii games Connect Four or Othello. The central research question is: *Are general-purpose RL agents (with pre-training) stronger than general-game-playing agents (with no pre-training)?*

The general Ludii interface for GBG is already there, see [Weitz2022a] & [Scheiermann2020].

This project builds upon the completed project Ludii interface for GBG, but extends it to the not yet covered games Nim, Hex and Connect Four. It makes a larger survey on different agent configurations.

Work steps:

- Get familiar with the Ludii interface as described in [Weitz2022a] and implemented in GBG.
- Extend the interface, which currently supports Hex, Othello, Yavalath, to support also the new games Nim and Connect Four. Needs a translation of GBG states and actions to Ludii states and actions.
- Build RL Java agents according to the Ludii interface. “Inside” they should host the proper GBG agents. Possibilities: TD n-tuple or Sarsa n-tuple (both pre-trained in GBG).
- Make a competition of these agents against the various Ludii agents on the specific games Nim, Hex, and Connect Four.
- Nim offers a special option for research: It is a scalable game (the number and the size of the heaps can be varied), being probably more complex for self-trained agents in the case of larger heaps. Measure the performance of Ludii agents as a function of number and/or size of heaps.

Prior knowledge:

- Java
- Knowledge in RL helps, but can be also acquired on-the-job

PERFECT- OR STRONG-PLAYING AGENTS FOR LUDII

For some games there exist perfect- or strong-playing agents (programmed, not self-learned). Some of them are included in GBG and used for evaluation purposes against the self-learning agents. The perfect- or strong-playing agents are:

- Nim: Bouton’s agent
- Connect Four: Thill’s Alpha-Beta agent with opening book
- Othello: Edax (https://github.com/abulmo/edax-reversi by Richard Delorme), available in GBG

Ludii (https://ludii.games/) is an interesting game platform with many games and with a number of different tree-based agents (MCTS, Alpha-Beta, no pre-training). It offers the possibility to play with a user-supplied JAR-agent against the Ludii agents. Ludii does not offer the possibility to play against perfect- or strong-playing game-specific agents. It would be interesting to see, how the above agents compare with the built-in Ludii agents on the Ludii games Nim, Connect Four and Othello. The central research question is: *How far from perfect or strong playing (in the context of certain games) are the general-game-playing agents in Ludii?*

The general Ludii interface for GBG is already there, see [Weitz2022a] & [Scheiermann2020]

Work steps:

- Get familiar with the Ludii interface as described in [Weitz2022a] and implemented in GBG.
- Extend the interface, which currently supports Hex, Othello, Yavalath, to support also the new games Nim and Connect Four. Needs a translation of GBG states and actions to Ludii states and actions.
- Build Java agents according to the Ludii interface. “Inside” they should host the proper GBG agents: Nim – Bouton, Connect Four – Alpha-Beta, Othello – Edax.
- Make a competition of these agents against the various Ludii agents on the specific games Nim, Connect Four or Othello.

- Nim offers a special option for research: It is a scalable game (the number and the size of the heaps can be varied), being probably more complex for general-game-playing agents in the case of larger heaps. Measure the performance of Ludii agents as a function of number and/or size of heaps.

Prior knowledge:

- Java

RUBIK'S CUBE

This project has been partly done by own work (see [Scheier2022a], local copy and [Konen2023a] , local copy), but **perhaps extensions are possible in one way or the other**.

Rubik's cube (https://de.wikipedia.org/wiki/Zauberw%C3%BCrfel) is so famous that it probably does not need to be introduced. The state space complexity 4.3*10^{19} is enormous.

Can a computer program **learn** to solve Rubik's cube? – Ideas might be time-reverse reinforcement learning, exploitation of color symmetries. Relatively open and complex task. There exist some recent research papers ([McAleer2018] = https://arxiv.org/pdf/1805.07470.pdf, [Agostinelli2019]) which claim that they have solved it, but the paper is not very clear about all the details. It might be an interesting task to check if their algorithm is understandable and to see if one can reproduce their results in GBG.

Even more interesting is the paper [Agostinelli2019], which has solved various cube aspects.

Current status:

- The 2x2x2 cube is now basically solved in GBG, but in [Scheier2022a] the 3x3x3 cube can be solved only for up to 8 scrambling twists. Recently, with the usage of color symmetries and other enhancements, this could be extended to up to 15 scrambling twists (QTM cube) [Konen2023a]. Nevertheless, a full solution 3x3x3x in GBG is still open (because God's number for 3x3x3 QTM cube is 26). Likewise, the 3D cube visualization is still open

ALPHAGO-ZERO-INSPIRED GBG AGENT

This project has already been done (Bachelor thesis Johannes Scheiermann 2020), but **perhaps extensions are possible in one way or the other**. See also [Scheier2022a].

AlphaGo Zero (Silver et al, 2017, *Nature*) (local copy) is probably the world's strongest Go player and also capable of learning other complex games. It is a clever combination of MCTS (Monte Carlo Tree Search) and DNN (Deep Neural Networks) which learns *tabula rasa* (from zero) and solely from self-play.

GBG is a General Board Game playing and learning framework which includes also agents learning solely from self-play. While the complex CNN-based deep neural networks in AlphaGo Zero are beyond reach for simple hardware, the combination "MCTS and neural network" can be carried over to GBG. This might lead already to big improvements when playing games where a certain thinking time is available.

An interesting point to research: Can such an agent beat the strong-playing Othello agent Edax on some higher Edax levels? Currently, the best TD agents are on par with Edax level-1, but they will consistently lose against Edax level-2 and higher, where 21 (!) Edax levels are available.

Interesting side project: There exists a very good tutorial on AlphaGo Zero:

- https://web.stanford.edu/~surag/posts/alphazero.html : Surag Nair, 2017, with pseudo code.

- https://github.com/suragnair/alpha-zero-general: A GitHub with full implementation in Python.

Work steps:

- Understand GBG's TD n-tuple agent
- Build an AlphaGo-Zero-inspired agent, where MCTS is coupled with a (pre-trained) TD n-tuple network (instead of a DNN).
- Evaluate the new agent with various parameters on various games and compare it with plain TD n-tuple agent and with n-ply wrapped TD n-tuple agent. Evaluation criteria: computation time, win rate.
- Make a special evaluation {Othello, Edax}.
- Optional: Integrate and test the idea from [Silver2017] that the search tree of MCTS is re-used in subsequent time steps of a game episode (re-using the subtree from the relevant child node which becomes the new root node).

Prior knowledge:

- Java
- MCTS and RL knowledge helps, but can be also acquired on-the-job

LUDII INTERFACE FOR GBG

This project has already been done (BA Johannes Scheiermann 2020, PP Ann Weitz 2022), BA Meltem Seven 2023). The current Ludii interface supports the games Hex, Othello and Yavalath, and – since the completion of BA Meltem Seven – also the games Nim and Connect Four. **Perhaps extensions are possible in one way or the other**.

How strong are GBG agents compared to other general-game-playing agents?

Ludii (https://ludii.games/) is an interesting game platform, developed by Cameron Browne and his team, with a large variety of games and with a number of different tree-based agents (MCTS, Alpha-Beta, no pre-training). It offers the possibility to play with a user-supplied JAR-agent against the Ludii agents. Ludii does not offer the possibility to train **reinforcement learning (RL) agents**. It would be interesting to see, how RL agents for Connect Four or Othello, that were trained in GBG and packed into a JAR, compare with the built-in Ludii agents on the Ludii games Connect Four or Othello. The central research question is: Are general-purpose RL agents (with pre-training) stronger than general-game-playing agents (with no pre-training)?

Work steps:

- Start with a simple Java GBG agent (Random or MCTS) and wrap it into a Ludii JAR in order to get familiar with the Ludii interface.
- Build RL Java agents according to the Ludii interface. "Inside" they should host the proper GBG agents. Possibilities: TD n-tuple or Sarsa n-tuple (both pre-trained in GBG). Needs a translation of GBG states and actions to Ludii states and actions.
- Make a competition of these agents against the various Ludii agents on the specific games Hex, Othello, Yavalath.

Prior knowledge:

- Java
- Knowledge in RL helps, but can be also acquired on-the-job

OTHELLO VS. EDAX

This project has already been done (Bachelor thesis Johannes Scheiermann 2020) and was extended in the journal paper by Scheiermann and Konen [Scheier2022a]. But **perhaps extensions are possible in one way or the other**.

Othello, https://en.wikipedia.org/wiki/Reversi, a medium-complexity game, is interesting, because it is still unsolved.

Othello is now available in GBG. It comes with a very strong playing Othello-specific agent Edax (https://github.com/abulmo/edax-reversi by Richard Delorme). Although Othello is available, there is need for further research. All Edax-levels above level 1 are very tough to beat. Currently, none of the generic agents (trainable RL-agents or MCTS agents) can beat Edax beyond level 1. Can we improve on that? Can we design an agent trainable by self-play which reliably beats Edax at some of the higher levels 2, 3, 4, …,15? – Might be tackled via **AlphaZero-inspired GBG agent**.

COMPETITION FRAMEWORK FOR GBG

This project has already been done (Master thesis Felix Barsnick 2019), but **perhaps extensions are possible in one way or the other**. The Tournament System has for example not yet been tested thoroughly on different games.

From ToG-GBG.pdf:

> *A game **competition** is a contest between agents. There are multiple objectives which play a role in such a contest:*
> *a) ability to win a game episode or a set of game episodes, maybe from different start positions or against different opponents (results clearly depend on the nature of the other agents);*
> *b) ability to win in several games;*
> *c) time needed during game play (either time per move or budget per episode or per tournament);*
> *d) time needed for setting up the agent (training), and possibly other objectives in training.*
> *Other objectives are possible as well.*
>
> *Competition objectives may be combined, i. e. how is the agent's ability to win if there is a constraint on the play time budget. When running this with different budgets, a curve 'win-rate vs. time budget' can be obtained.*
>
> *Methods from multi-objective optimization (e. g. Pareto dominance) can be used to inspect and visualize competition results.*

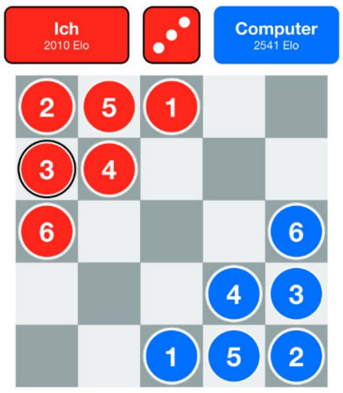
This leads to the following goals for a thesis project *Competition Framework for GBG* or *Tournament System*:

- Screen existing literature on game competitions
- Make a concept for a competition framework in GBG
- In the case of 1-player games, set up a competition with this rule: Each agent plays one or many episodes alone and the agent achieving the highest overall score wins.
- Round-robin tournaments:
  - selection of agents → GUI
  - result reporting: rank tables vs. score tables
- Include the time aspects:
  - integrate time measurement of agents
  - design a win-rate as a function of time constraints curve
- Multi-objective: Pareto plots: Win-rate vs. time
- Visualization of game-play during competitions
- Elo- and Glicko-rating
- Analyze the competition rules and competition results for existing games and agents: Are the competitions ‘fair’? Are there signs for non-transitivity? If so, how to establish a fair winner then?

EWN (EINSTEIN WÜRFELT NICHT)

This project has already been done (Bachelor thesis Julian Cöln 2022), but **perhaps extensions are possible in one way or the other**.

EWN, EinStein würfelt nicht!
(https://de.wikipedia.org/wiki/EinStein_w%C3%BCrfelt_nicht,[1]): EWN is a quite interesting game, not so easy, not too complex, non-deterministic, but nevertheless strategic. It is said that there exist 3- and 4-player-variants with teams (needs to be researched).



Work steps:

- Implement the basic 2-player version of EWN in GBG
- Research literature on 3- and 4-player variants
- Test various agents: Which agent with which parameters works best on EWN?
- Optional: Implement and test 3- and 4-player variants

Literature:

- http://hgm.nubati.net/Einstein.html

REALIZED GAMES

- ConnectFour [Thill14, Bagh14], which contains a perfect-playing AlphaBeta agent and a strong-playing TD-n-tuple agent, was ported to GBG.
- Sim, https://en.wikipedia.org/wiki/Sim_(pencil_game) (2-player variant, 3-player variant), an interesting game for N≥2 players. See also Learnability of Sim in GBG.
- Nim (2 players), Nim3P (3 players). See also Learnability of Nim/Nim3P in GBG.
- Othello (Reversi) [INF-Project Dittmar & Cöln]. See also Othello vs. Edax and AlphaGo-Zero-inspired GBG Agent.
- Hex [BA Kevin Galitzki]. See also CNN-like Agent.
- EWN (EinStein Würfelt Nicht!) [BA Julian Cöln].
- Yavalath [BA Ann Weitz]. See also Yavalath Extensions.
- Poker, Kuhn Poker [MA Tim Zeh]
- BlackJack [BA Simon Meissner]
- Rubik's Cube (beta version)
- 2048 [BA Johannes Kutsch].
- TicTacToe

LITERATURE

[Agostinelli2019] Agostinelli, Forest, et al. "Solving the Rubik's cube with deep reinforcement learning and search." Nature Machine Intelligence 1.8 (2019): 356-363.
(https://www.ics.uci.edu/~fagostin/assets/files/SolvingTheRubiksCubeWithDeepReinforcementLearningAndSearch_Final.pdf, **local copy**)

[1] See also http://computerschach.de/Files/2005/EinSteinen%20in%20Jenas%20langer%20Nacht.pdf: nice article with background info on EWN and commented game episode – in German.

[Fabig2021] Fabig, Niklas: „Portierung einer Java-Reaktorsimulation nach Python und Performanz-Vergleich mit verschiedenen Python-Java-Bridges“, TH Köln -- University of Applied Sciences (2021) https://www.gm.fh-koeln.de/~konen/research/PaperPDF/Projektarbeit_Niklas_Fabig-2021.pdf

[Konen2020b] W. Konen, S. Bagheri: „Reinforcement Learning for N-Player Games: The Importance of Final Adaptation”, BIOMA’2020, Brussels (2020) (preprint available here: http://www.gm.fh-koeln.de/ciopwebpub/Konen20b.d/bioma20-TDNTuple.pdf or here: somewhat longer technical report)

[Konen2023a] Konen, Wolfgang: “Towards Learning Rubik’s Cube with N-tuple-based Reinforcement Learning”, Technical Report, TH Köln, 2023. http://www.gm.fh-koeln.de/ciopwebpub/Konen22b.d/TR-Rubiks.pdf

[McAleer2018] McAleer S., Agostinelli F. et al.: “Solving the Rubik’s Cube Without Human Knowledge”, *arXiv preprint arXiv:1805.07470* (2018) (https://arxiv.org/abs/1805.07470, local copy)

[McAleer2019] McAleer S., Agostinelli F. et al.: “Solving the Rubik’s Cube With Approximate Policy Iteration”, *Int. Conf on Learning Representations (ICLR)* (2018) (https://arxiv.org/abs/1805.07470, local copy)

[Scheiermann2020] Scheiermann, Johannes: „AlphaZero-inspirierte KI-Agenten im General Board Game Playing“, Bachelor thesis, TH Köln -- University of Applied Sciences, 2020 http://www.gm.fh-koeln.de/~konen/research/PaperPDF/BA_Scheiermann_final.pdf

[Scheier2022a] Scheiermann, Johannes; Konen, Wolfgang: „AlphaZero-Inspired Game Learning: Faster Training by Using MCTS Only at Test Time“, IEEE Transactions on Games, 2022. doi:10.1109/TG.2022.3206733. Arxiv preprint available under https://arxiv.org/abs/2204.13307.

[Seven2023] Seven, Meltem: „KI-Agenten im Vergleich: Erfolg von Agenten des Ludii General Game Systems in Partien gegen perfekte oder starke Agenten am Beispiel der Spiele Vier Gewinnt, Nim und Othello“, Bachelor Thesis, TH Köln -- University of Applied Sciences, , 2023 https://www.gm.fh-koeln.de/~konen/research/PaperPDF/BA_Meltem-Seven-final.pdf

[Weitz2022a] Weitz, Ann: „Entwicklung einer allgemeinen Schnittstelle zwischen Ludii und dem GBG Framework“, Technical report, TH Köln -- University of Applied Sciences, , 2022 https://www.gm.fh-koeln.de/~konen/research/PaperPDF/PP-Doku-Weitz-2022-02.pdf

[Weitz2022b] Weitz, Ann: „Untersuchung von selbstlernenden Reinforcement Learning Agenten im computergenerierten Spiel Yavalath“, Bachelor Thesis, TH Köln -- University of Applied Sciences, , 2022 https://www.gm.fh-koeln.de/~konen/research/PaperPDF/BA-Weitz-final-2022.pdf