Ceramic: A research environment based on the multi-player strategic board game Azul

Quentin Gendre[1,a)] Tomoyuki Kaneko[2,b)]

Abstract: Strategy board games such as Chess, Shogi and Go are popular benchmark for Artificial Intelligences (AIs). However, there already exist very strong agents that outperform humans for these games. In order to expand AI research, it is necessary to turn to new difficult strategy games, such as with more than 2 players. This paper presents Ceramic, a new environment based on the board game Azul with such challenge, as well as randomness, a large state & action space and perfect information. In a game of Ceramic, 2 to 4 players take turns in selecting tiles from a common shared set of *factories*, and placing them on their *pyramid* and *wall* to form patterns and score points. In order to have a strong strategy, a player must plan his moves as well as anticipate the opponents'. This paper presents the framework's various functionalities, and provides an in-depth analysis of the different baseline players implemented. The strongest AI, *mc*-10 000*, significantly out-performs the others, but remains weaker than human players, indicating the game is difficult for AIs and there is a large margin of improvement. The code is open-source, and was binded into a python module to provide a simple interface to python libraries, and thus ease the training and evaluation of Reinforcement Learning-based AIs.

Keywords: Reinforcement Learning framework, Board game, Multi-agent, Monte Carlo, UCT

1. Introduction

Games, and more specifically board games, have for long served as testbeds for Artificial Intelligence (AI) research. Since the success of Deep Blue at Chess in 1997 [1], researchers have moved on to create AI agents that can outperform humans in more and more difficult environments: AlphaGo and AlphaGoZero at Go in 2015 and 2017 [8], AlphaZero at Shogi, Chess, and Go [9], and AlphaStar at Starcraft II in 2019 [10].

To push AI's potential further, it is necessary to create new research environments, with new challenges that haven't been solved yet. Some frameworks focus on playing consistently on multiple games, for example the General Video Game AI Framework [7]. Others focus on a complex game, such as the 3D environment of Minecraft (Malmo [3]), or the strategic collectible card game Hearthstone (The Hearthstone AI framework [2]).

One important challenge is multi-agent systems. In real-life scenarios, agents are rarely alone and share the environment with other entities. A sub-problem of this is multiplayer games with 3 or more players, which are a great representatives of such domains. Catan, with the Java implementation JSettlers [6], is a modern board game example.

In this paper, we introduce Ceramic, an environment and research framework inspired by the board game Azul, itself influenced by the Portuguese ceramic tiles *azulejos*. Azul is a two to four player strategy board designed by Michael Kiesling in 2017 that won the Spiel des Jahres award (Game of the Year) the next year. In the game, players take turns in picking tiles from a shared set of *factories*, and forming patterns on their *pyramid*. It has very interesting multi-player mechanics, where one player must take into accounts the tiles picked by the opponents in order to always have access to good tiles, not get backed into a corner and instead force opponents to take bad actions. Players must also plan their tiles placement to maximize their score, without taking too much risks and getting stuck later in the game.

The source code of Ceramic is open-source (under the GPL-3.0 license), and available at the following address: `https://github.com/Swynfel/ceramic`. It has been programmed in C++ for its efficiency, but we also included bindings to turn it into a python module, thus it is easily usable in combination with PyTorch, Tensorflow, or other

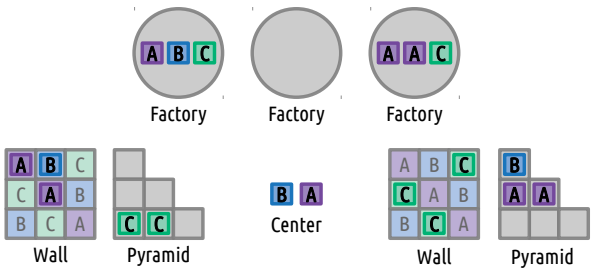

Fig. 1 Example of 2-player *mini* ceramic state (bag, bin, and floor are not shown)

[1] Graduate School of Interdisciplinary Information Studies, the University of Tokyo
[2] Interfaculty Initiative in Information Studies, the University of Tokyo
[a)] gendre@game.c.u-tokyo.ac.jp
[b)] kaneko@graco.c.u-tokyo.ac.jp

python libraries. We welcome researchers interested in making agents for this game to re-use our code.

2. Game of Ceramic

The rules of Ceramic basically follows Azul's[*1], but supports rule variations to allow a greater range of environments. In the game, players pick tiles in *factories*, forming patterns in a *pyramid* and score points by placing them on a *wall*. Ceramic is very challenging, as good players must anticipate what tiles opponents will take (to potentially hinder them), as well as balancing short-term and long-term decisions, risk-taking and safely scoring.

The rules of Ceramic are configurable by several parameters, $C, T, F, N, S_v, S_h, S_c$, with typical combinations listed in Table 1. In the rest of the paper, if no rules are specified, the *base*[*1] rules are implied.

Table 1 Possible rule parameters

	C	T	F	N	S_v	S_h	S_c
Base	5	20	$1+2\times p$	4	2	7	10
Mini	3	15	$1+p$	3	1	4	6

2.1 Rules

The game is played in multiple rounds. In each round, players will collect tiles to ultimately place on their wall to score points. The player that ends with the highest score wins.

Let $p \in \{2,3,4\}$ be the number of players. There are C colors with T tiles of each color, initially placed in a *bag*. Each player has their own *Wall* with $C \times C$ cells, C-base *Pyramid*, *Floor* (i.e. penalty) and Score.

2.1.1 Round setup

F *factories* are placed in the middle, each filled with N random tiles. If no tiles are left in the *bag*, discarded tiles in the *bin* are moved to the *bag*. If no more tiles remain in the *bin* as well, factories are left empty or half-filled and the game continues.

The player with the first player token goes first, after placing the token back in the *center*. At the very start of the game, there is no token, so the player order is determined randomly.

2.1.2 Acquiring tiles

At their turn, a player chooses a *factory*, a tile color on it, and takes all the tiles of this color. The factory's tiles of a different color are moved to the *center*. Instead of selecting a *factory*, a player may also choose to take all tiles of the same color from the *center*.

Once a player has picked tiles, they may place them on a line of their *pyramid*, as long as certain conditions are met: a *pyramid*'s line must only hold tiles of the same color, and this color must not be present on the wall's matching line. Extra tiles that can't fit – or all of them if the player cannot/doesn't want to choose a line – are placed on the player's *floor*, i.e. penalty line.

If the player took tiles from the *center*, and is the first one to do so this round, they must also take the first player token and place it on their *floor*.

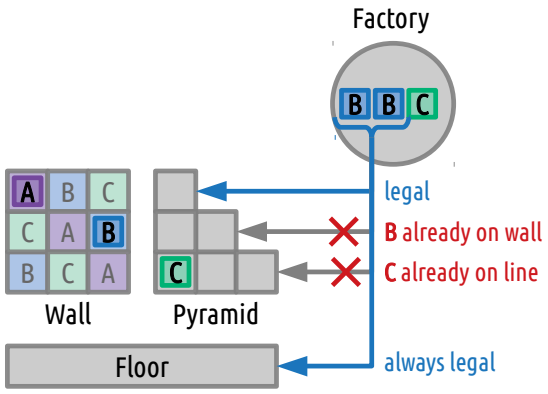

Fig. 2 Example of where tiles "B" of *factory* can be placed

After a player completes their turn, the next player in clockwise continues, and so until no tile remains on the *factories* or *center*.

2.1.3 Round end and Scoring

At the end of the round (c.f. above), the following should be done for each player: For each *filled* line on the *pyramid* (from smallest to largest line in case multiple are filled) one of the tiles moves to the wall at the corresponding height and color, and the rest is discarded to the *bin* (c.f. Fig. 3). When a tile is placed on a *wall*, it scores a point for each consecutive horizontal or vertical tile, or a single point if isolated (c.f. Fig. 4).

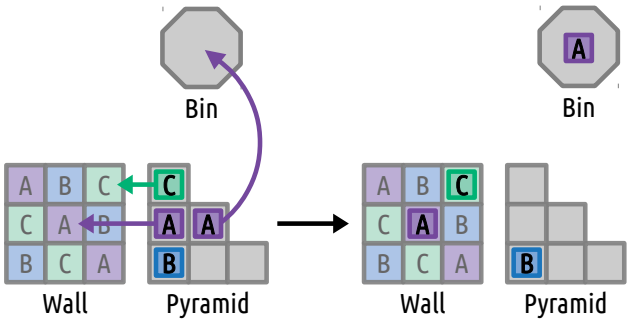

Fig. 3 Example of tile placement from pyramid to wall

Tiles on the *floor* decrease points, then they are sent to the *bin*. The tiles on the *pyramid* or *wall* are left untouched.

2.1.4 End of game

If, at the end of a scoring round, a player has a line on their wall completed, the game stops. To prevent soft-locking, the game is also stopped if no more tiles of a certain color are left. Extra points are given for each complete vertical line S_v, horizontal column S_h, and color set S_c on the walls.

The player with the highest score wins the game. In case of tie, the one with the largest amount of filled horizontal lines wins. In case of another tie, the last player to have played their first move wins.

[*1] Ceramic's *base* rules are identical to Azul's, albeit the equality break for winner and the ending condition when a tile color is depleted

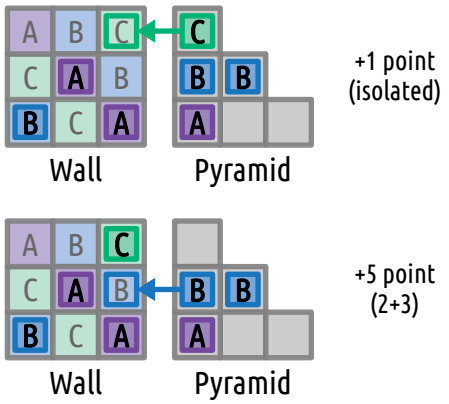

Fig. 4 Example of score obtained during placement

2.2 Challenges and Specificities

Ceramic is a perfect information game (i.e. no information is hidden), and – although the Round setups are random – every action within a round has a deterministic outcome. This means that even though Ceramic is very difficult, it still follows strong conditions. This enables many existing algorithms (such as Monte Carlo Tree Search) to play the game without cheating.

However the game tree is very large and deep. To take the 4-player *base* rules as an example, finishing a game requires at least 5 rounds, each consisting of 10 to 36 moves. The number of legal actions decreases throughout the rounds, but can start at 228: 36 tile sets (selection) times 5 pyramid lines plus 1 penalty line (placement). Combined with the random factory refill between rounds, this makes perfectly solving the game with current algorithms (e.g. preferences of outcomes) impossible.

Furthermore, the game is multi-player, meaning a strong agents must take into consideration not only its own panel and actions, but also the opponents' and their interactions between them. Not only should an eye be kept on the leading player, but it is necessary to predict what the opponents will play in order to avoid being stuck in a situation where one is forced to play a bad action. For example, it is common at the end of the round that the *center* contains a lot of tiles that have accumulated throughout the round, and unprepared players forced to take them, place them on the *floor* and lose a lot of points

Finally, the interpretation of the state is very complex. In the *base* rules, the features need to represent a combination of one panel for each player (composed of a 5 by 5 *wall* and a *pyramid* of size 5), and a common board (composed of a *center* and 5 to 9 *factories* that should be invariant by permutation). This makes it rather challenging for AI based on neural networks.

3. Related Work

As Ceramic is a new environment, there exists no preexisting work for this specific game. However, solving similar problems is common, so we were able to base our work on the following.

3.1 Markov Decision Process (MDP)

A game in Ceramic can be represented as a Markov Decision Process (MDP). Namely, when at a state s, an agent can take an action a following policy π, and will receive a reward r and the next state s'.

Opponents can either be seen as part of the environment (but it ignores the multi-agent dynamics), or the definition can be extended to a Markov Game [5], where each player takes their own actions.

There are multiple ways to define the reward in Ceramic: as the delta of the guarantied score after every move, the delta of the raw score at every round, or simply winning at the very end. The final score of a game (i.e. cumulative reward) is called the return.

3.2 Monte-Carlo Tree Search

Monte-Carlo Planning is a very popular technique to select actions in complex MDP. This idea is to approximate the relevance of an action with roll-outs, i.e. simulating it and finishing the game randomly multiple times, and consider their average outcome as a good approximation of the expected outcome.

3.3 Upper Confidence Bounds applied to Trees (UCT)

Upper Confidence Bounds applied to Trees (UCT) is a common technique to improve Monte-Carlo's balance between exploration and exploitation [4]. Instead on exploring branches randomly, since ones with poor outcome are unlikely to be played anyway, branches with the best outcome are prioritized.

At every state s already encountered N times, we have for each possible action a, already encountered n_a times and with an average return of X_a, the next sampled action a' will be chosen as:

$$a' = \underset{a}{\operatorname{argmax}} \left(\bar{X}_a + c\sqrt{\frac{\ln N}{n_a}} \right)$$

with c a constant, often given the value $\sqrt{2}$.

4. Framework

As mentioned in the introduction, we implemented an optimized framework for playing Ceramic games, that we also called Ceramic. It is open sourced under the GPL-3.0 license, and available on github at the following address: `https://github.com/Swynfel/ceramic`. Additional help on how to use the framework can be found there.

The code is written mostly in C++ for its execution speed, but bindings in pybind11 enables it to be compiled as a python module, thus providing a simple interface for python Machine Learning libraries such as Tensorflow and PyTorch.

Ceramic has many features useful for researchers wanting to conduct Reinforcement-Learning research, such as:

- Flexible rules and parameterization. This can be useful to simplify or play a variation of the game. For example, if one wants a minimal environment to test their

agents, we recommend using the *mini* rule-set introduced in Table 1.
- Fast concurrent execution, to allow quick experience generation and evaluation (c.f. Section 6.1). For example, an "arena" module for evaluating and comparing multiple agents playing against each others.
- Multiple baseline-agents, that can play in any rule-set (c.f. Section 5).
- Helper functions and an abstract base player class, to quickly prototype an agent in C++ or python.
- A "terminal-player", i.e. an interface in the command line so humans can play against the different agents.

5. Agents

5.1 First Legal Player

This "First Legal" Player is deterministic and takes the first legal action it finds. Actions are ordered from longest *pyramid* line to shortest, then *center* or *factory* id, and finally color. This is the simplest and fastest agent, but it plays poorly. In the rest of the paper, this agent's name will be abbreviated as "*f-legal*".

5.2 Random Players

The random agent (abbreviated as "*rand-naive*") takes a random action among the available legal actions. It is also very fast, and has the feature of being stochastic.

Placing tiles directly on the *floor* is almost always bad, as it decreases score without making progress on the *pyramid* or *wall*. However, theses actions can represent a significant proportion of legal actions. Thus, we added a "smart" variation of the random agent that prioritizes actions placing tiles on the *pyramid*, abbreviated as "*rand-smart*". This new agent plays significantly better than the "naive" random agent (c.f. Section 6.2) while taking the same amount of time (c.f. Section 6.1).

5.3 Monte-Carlo Players

The Monte-Carlo player is based on Monte-Carlo tree search and UCT described in Section 3. The first action to explore is chosen according to UCT (with $c = \sqrt{2}$), and the rest of the roll-out continues as if played by *rand-smart* (i.e. "smart" random players).

Since the game tree can be a little deep, we implemented two alternatives:
- One that stops roll-outs after the end of the round, and estimates the win-rate with a hand-coded heuristic that takes into account its score and its wall configuration relative to the other players.
- Another one that plays roll-outs until the end of the game, and returns 1 if the player won and 0 otherwise.

These two agents are respectively abbreviated as "*mc-n*" and "*mc*-n*", with n the number of roll-outs. The higher the amount of roll-outs, the stronger the Monte-Carlo Players become, at the expense of more processing time.

6. Experiments and Results

All experience were conducted one at a time, on a server with an AMD Ryzen Threadripper 2990WX (a 3GHz, 32-core/64 processes CPU) and 64 GB of RAM.

6.1 Environment Bench-marking

The first set of experiences focuses on the performance of the environment. For Figure 5, 10 *rand-smart* agents played in an arena in "all" groups (which amount to 705 4-player groups, see Section 6.2 for explanations) with 1000 games per group (705 000 games in total). The values in Table 2 were collected during the experiments of Section 6.2: un-marked values correspond to measurements made during the experiments of Table 4, and those marked with † to the ones of Table 3.

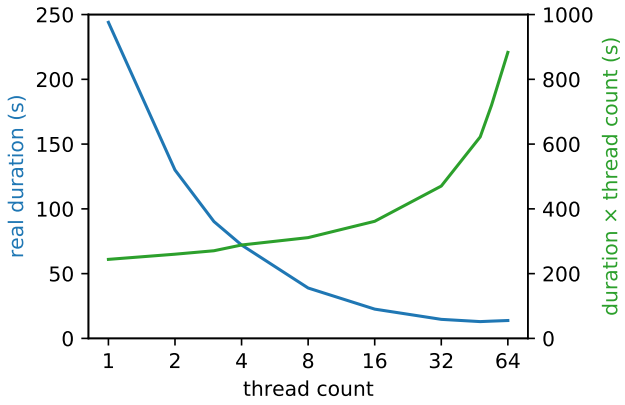

Fig. 5 Execution time of 10-*rand-smart* 1000-games 4-player "all" arena depending on the number of threads

Table 2 Average processing times

	duration (in μs)
Play full game	4.944×10^7
† Play full game	4.900×10^2
Process action	2.700×10^0
† Process action	9.588×10^{-1}
† *f-legal*	3.047×10^0
† *rand-naive*	4.731×10^0
† *rand-smart*	4.602×10^0
rand-naive	7.399×10^0
rand-smart	6.837×10^0
mc-100	7.646×10^3
mc-1 000	7.543×10^4
mc-10 000	7.207×10^5
mc-100*	3.876×10^4
mc-1 000*	3.821×10^5
mc-10 000*	3.387×10^6

We can see on Table 2 that the environment is very fast to process an action (less than $10\mu s$), and that most of the time is taken by the agents making a decision. Unsurprisingly, the Monte-Carlo players are the slowest to play, taking around seconds to play with 10 000 samples. We decided to not go further than this, as it would be too slow to actually use for training or evaluation.

6.2 Performance between agents

The following two tables compare the different agents' performance by opposing them in several groups, each group playing a certain amount of games, and taking the average win-rate, score, and moves for each player. We call this a n-game "*type*" p-player arena, with n the number of games for each group, and "*type*" a value that can be either "subsets" or "all". A "subsets" arena takes all p-subsets of the available players as groups, and the "all" arena takes all p-combinations allowing copies of the same agent (but with at least two different types). [*2]

Since the games are 4-players, the average win-rate is 25%. The column *sstd* stands for "score standard deviation".

Table 3 Results of simplest agents in a 100 000-game 4-player "all" arena
Each agents played 1 600 000 games

	win-rate	score	sstd	moves
f-legal	27.54%	20.0	11.2	25.7
rand-naive	2.39%	3.6	5.7	24.2
rand-smart	45.07%	25.0	13.8	23.0

Table 4 Results of various agents in a 1000-game 4-player "subsets" arena
Each agents played 35 000 games

	win-rate	score	sstd	moves
rand-naive	0.01%	2.4	4.4	21.3
rand-smart	3.25%	18.6	11.7	21.1
mc-100	9.84%	21.0	13.6	21.2
mc-1 000	15.04%	23.5	15.1	21.2
mc-10 000	16.59%	24.9	15.7	21.2
mc-100*	13.55%	27.1	15.3	21.0
mc-1 000*	49.67%	46.8	19.2	21.1
mc-10 000*	92.05%	67.7	17.2	21.5

As it can be seen on Table 4, *mc** are clearly the strongest agents, *mc*-10 000* even reaching *92.05%* win-rate. However, as mentioned in Section 6.1, they are also the agents that take the longest time to play: *mc*-10 000* takes more than 3 seconds on average (c.f. Table 2). This means that *mc* agents with less roll-outs or the smart random agent *rand-smart* might be better alternatives in situations where experiences must be generated quickly.

6.3 Performance against humans

The agent *mc*-10 000* seems very strong in comparison to the other players. However, we do not know about its performance against human players. For this, one of us (Quentin Gendre) played 50 games against 3 *mc*-10 000* agents. The results are given in Table 5.

Although 50 games is too few to give a precise win-rate, the difference of games won and score is significant enough to conclude that humans still out-perform *mc*-10 000*, and thus all currently available AI-players.

[*2] For example, with $p = 3$, if players A, B, C and D are available: "subsets" arena would generate groups ABC, ABD, ACD, and BCD; whereas "all" arena would generate groups AAB, AAC, AAD, ABB, ABC, ABD, ACC, ACD, ADD, BBC, BBD, BCC, BCD, BDD, CCD, and CDD

Table 5 Results of 50 games of human vs 3 *mc*-10 000*

	games won	score
human	45	98.2
3×*mc*-10 000*	5	54.2

games won by	score	score of 2nd player
human	101.7	65.8
mc-10 000*	74.6	70.0

6.4 Statistics

Finally, in order to have various statistics on a typical game of Ceramic, we had 4 *mc*-10 000* play 10 000 games and analyzed their behavior.

Table 6 Statistics of 10 000 games of 4 *mc*-10 000*

	average
Round per game	5.06
Moves per game	82.2
Penalty per round	3.51
Legal actions	44.7
Smart legal actions	31.6
Tiles taken	2.22
Score of winner	63.6
Score of 2nd player	50.4
Score of 3rd player	36.5
Score of last player	21.9
Tiles on wall	13.0
Filled lines	0.625
Filled columns	0.165
Filled colors	0.384

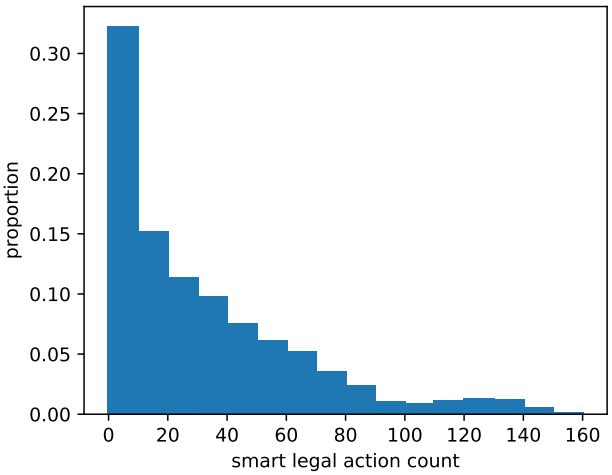

Fig. 6 Distribution of smart legal action count in 10 000 games of 4 *mc*-10 000*

We can see in Table 6 that, on average, a player will have access to 44.7 legal actions, or 31.6 actions if we ignore the ones that place the tiles directly in the *floor* when placing them on the *pyramid* is possible. However, agents may have to choose among larger sets. Empirically, *mc*-10 000* has more than 5% chance to encounter more that 100 smart legal actions (c.f. Fig. 6).

Furthermore, a game with good players will almost always end in 5 rounds, the minimum, meaning players are able to place tiles on the same line at each round without interruption.

We can note that *mc*-10 000* rarely use the filled line/column/color bonuses: on average, they have only around one.

7. Conclusion and Future Works

In this paper we present Ceramic, a game environment and framework for Artificial Intelligence research. A game of Ceramic has interesting multi-player dynamics that AIs should properly take into account to maximize their win-rate. We implemented several base-line players, ranging from very fast ($< 10\mu s$ per action) random agents to slower but stronger Monte-Carlo agents.

Although the best player implemented, *mc*-10 000*, is significantly better than the random agents, it is still very weak in comparison to humans. It illustrates how making agents that play Ceramic is very challenging, because of multi-player, randomness, and large game space. Therefore, we strongly believe doing research on this environment would give valuable insights on real world problems with similar difficulties. Ceramic is open-source to encourage such initiatives, and we hope to see researchers make RL-based agents for it.

References

[1] Campbell, M., Hoane Jr, A. J. and Hsu, F.-h.: Deep blue, *Artificial intelligence*, Vol. 134, No. 1-2, pp. 57–83 (2002).

[2] Dockhorn, A. and Mostaghim, S.: Introducing the Hearthstone-AI Competition, *CoRR*, Vol. abs/1906.04238, pp. 1–4 (online), available from ⟨http://arxiv.org/abs/1906.04238⟩ (2019).

[3] Johnson, M., Hofmann, K., Hutton, T., Bignell, D. and Hofmann, K.: The Malmo Platform for Artificial Intelligence Experimentation, *25th International Joint Conference on Artificial Intelligence (IJCAI-16)*, AAAI - Association for the Advancement of Artificial Intelligence (2016).

[4] Kocsis, L. and Szepesvári, C.: Bandit Based Monte-Carlo Planning, *Machine Learning: ECML 2006* (Fürnkranz, J., Scheffer, T. and Spiliopoulou, M., eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 282–293 (2006).

[5] Littman, M. L.: Markov games as a framework for multi-agent reinforcement learning, *Machine learning proceedings 1994*, Elsevier, pp. 157–163 (1994).

[6] Monin, J. and Contributors: JSettlers2 release-2.2.00, `https://github.com/jdmonin/JSettlers2/` (2020).

[7] Perez-Liebana, D., Samothrakis, S., Togelius, J., Schaul, T. and Lucas, S.: General Video Game AI: Competition, Challenges and Opportunities, *AAAI Conference on Artificial Intelligence* (2016).

[8] Silver, D. et al.: Mastering the game of go without human knowledge, *Nature*, Vol. 550, No. 7676, pp. 354–359 (2017).

[9] Silver, D. et al.: A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play, *Science*, Vol. 362, No. 6419, pp. 1140–1144 (online), DOI: 10.1126/science.aar6404 (2018).

[10] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P. et al.: Grandmaster level in StarCraft II using multi-agent reinforcement learning, *Nature*, Vol. 575, No. 7782, pp. 350–354 (2019).